Space Efficient Algorithms for Graph Isomorphism and Representation

D I S S E R T A T I O N

zur Erlangung des akademischen Grades

Dr. rer. nat.
im Fach Informatik

eingereicht an der
Mathematisch-Naturwissenschaftlichen Fakultät
der Humboldt-Universität zu Berlin

von
Dipl.-Inf. Sebastian Kuhnert

Präsident der Humboldt-Universität zu Berlin:
Prof. Dr. Jan-Hendrik Olbertz

Dekan der Mathematisch-Naturwissenschaftlichen Fakultät:
Prof. Dr. Elmar Kulke

Gutachter:
1. Prof. Dr. Johannes Köbler
2. Prof. Dr. Nicole Schweikardt
3. Prof. Dr. Martin Grohe

Tag der Verteidigung: 2. Februar 2016

Abstract

The *graph isomorphism problem* deals with the question if two graphs have the same structure up to renaming their vertices. It is one of the few remaining natural problems for which neither a polynomial-time algorithm nor NP-hardness is known. This situation has led to a branch of research that develops efficient algorithms for special cases of the graph isomorphism problem, where the input graphs are required to be from restricted graph classes.

The main contribution of this thesis comprises of logspace algorithms that solve the isomorphism problem for k-trees, interval graphs, Helly circular-arc graphs and proper circular-arc graphs. This improves previously known parallel algorithms and leads to a complete classification of the complexity of these problems, as they are also shown to be hard for logspace.

In fact, these algorithms achieve more: In the case of k-trees, the algorithm computes *canonical labelings* in space $O(k \log n)$. An alternative implementation runs in time $O\big((k+1)!\, n\big)$, where n is the number of vertices, yielding the fastest known FPT algorithm for k-tree isomorphism.

The algorithms for interval and circular-arc graphs actually compute *canonical representations*, i.e., each vertex is assigned an interval (or arc) such that these intersect each other if and only if the corresponding vertices are adjacent, and isomorphic input graphs receive the same interval (or arc) model. This thesis also presents logspace algorithms that compute interval representations with additional properties, or detect that this is not possible: The resulting interval models can be required to be proper (no interval contains another), unit (all intervals have the same length), or to satisfy prescribed lengths for pairwise intersections (and possibly prescribed lengths of intervals).

Zusammenfassung

Beim *Graphisomorphieproblem* geht es um die Frage, ob zwei Graphen bis auf Knotenumbenennungen die gleiche Struktur haben. Es ist eines der wenigen verbleibenden natürlichen Probleme, für die weder ein Polynomialzeitalgorithmus noch NP-Härte bekannt ist. Aus dieser Situation ist ein Forschungszweig erwachsen, der effiziente Isomorphiealgorithmen für eingeschränkte Graphklassen entwickelt.

Der Hauptbeitrag dieser Arbeit besteht in Logspace-Algorithmen, die das Isomorphieproblem für k-Bäume, Intervallgraphen, sowie Helly- und Proper-Kreisbogengraphen lösen. Dies verbessert zuvor bekannte parallele Algorithmen und führt zu einer vollständigen Klassifikation der Komplexität dieser Probleme, da für sie auch Logspace-Härte nachgewiesen wird.

Tatsächlich leisten die vorgestellten Algorithmen mehr: Im Fall der k-Bäume berechnet der Algorithmus *kanonische Knotenbenennungen* mit $O(k \log n)$ Platz. Eine alternative Implementation des Algorithmus kommt mit $O\big((k+1)!\, n\big)$ Zeit aus – hierbei ist n die Anzahl der Knoten – und ist damit der schnellste bekannte FPT-Algorithmus für Isomorphie von k-Bäumen.

Die Algorithmen für Intervall- und Kreisbogengraphen berechnen *kanonische Repräsentationen* – das heißt, sie weisen jedem Knoten ein Intervall (beziehungsweise einen Kreisbogen) zu, sodass diese sich genau dann schneiden, wenn die zugehörigen Knoten benachbart sind, und isomorphe Eingabegraphen das gleiche Intervallmodell (beziehungsweise Kreisbogenmodell) erhalten. Außerdem werden auch Logspace-Algorithmen angegeben, die Intervallrepräsentationen mit zusätzlichen Eigenschaften berechnen – oder erkennen, dass dies nicht möglich ist: Für die resultierenden Intervallmodelle kann gefordert werden, dass sie *proper* sind (also kein Intervall ein anderes enthält), dass sie *unit* sind (also alle Intervalle die gleiche Länge haben) oder dass die Längen der paarweisen Schnitte (und optional der einzelnen Intervalle) vorgegebenen Werten entsprechen.

soli deo gloria

Acknowledgements

I thank my advisor Johannes Köbler for introducing me to the research area of Theoretical Computer Science and for his continued support and guidance. His expertise, encouragement, and perspective have proved invaluable throughout the work on this thesis.

I am deeply grateful for the inspiring discussions with him and my other coauthors: V. Arvind, Bireswar Das, Bastian Laubner, Gaurav Rattan, Jacobo Torán, Yadu Vasudev, Oleg Verbitsky, and Osamu Watanabe. They have made research a joyful experience for me. V. Arvind also hosted me several times at the Institute of Mathematical Sciences in Chennai, and I gladly remember these visits for the open atmosphere of the discussions and the fruitful collaboration.

I highly appreciate the comments of Frank Fuhlbrück, Manfred Kuhnert, Peter Patzt, and Stephan Verbücheln, who proofread drafts of this thesis.

Much of this work was supported by the DFG grants KO 1053/7–1 and –2. The visits to Chennai were supported by the Alexander von Humboldt Foundation in its institute partnership program.

Last but not least, I thank my parents. They have nourished my curiosity to discover, sparked my desire to understand how things work, and helped me develop the confidence it takes to complete a project like this.

Contents

1 Introduction

Two graphs G and H are *isomorphic* if there is a bijection φ between the vertex sets of G and H that preserves the adjacency relation, i.e., φ maps edges to edges and non-edges to non-edges. The *graph isomorphism problem (GI)* asks whether two given graphs are isomorphic. It is one of the few natural problems in NP that are neither known to be NP-complete nor known to be solvable in polynomial time. Trying to settle this question, many researchers have devoted their time to the graph isomorphism problem. Already in 1977, this phenomenon was called the *graph isomorphism disease* in the survey of Read and Corneil [RC77]. More surveys followed [Gat79; ZKT85; AT05; Köb06; Ueh14], witnessing the continued interest in GI.

It is known that GI is contained in coAM [GS86; Sch88] and in SPP [AK06]. This provides strong evidence that GI is not NP-complete, as that would imply a collapse of the polynomial hierarchy to its second level [BHZ87]. On the other hand, the strongest known hardness result is surprisingly weak; in particular, it is not known whether GI is hard for P. Torán showed that GI is at least as hard as computing the determinant of an integer matrix and thus hard for the class DET [Tor04]. DET is a subclass of TC^1 and contains NL as well as all logspace counting classes [ÁJ93; BDH+92]; see Figure 1.1 for an overview of the mentioned complexity classes. More details on the structural complexity of GI can be found in the monograph by Köbler, Schöning, and Torán [KST93].

The best known algorithm for GI takes $2^{O(\sqrt{n \log n})}$ time [BL83]; here and elsewhere, n denotes the number of vertices in the input graphs. This algorithm combines Luks' algorithm for bounded degree graphs [Luk82] with Zemlyachenko's techniques for degree reduction [ZKT82].

One line of research on GI is to study the complexity for particular classes of graphs. Two cases can be distinguished: *isomorphism-complete* graph classes, where the problem remains as hard as in general, and *isomorphism-tractable* graph classes, for which GI can be solved in polynomial time. Similarly as in the theory of NP-completeness, a dichotomic phenomenon can be observed: Almost all natural graph classes are known to be either isomorphism-complete or isomorphism-tractable; the most prominent exceptions are trapezoid graphs [Spi03; UTN05] and circular-arc graphs (see below). Paralleling Ladner's theorem, which asserts that NP contains languages that are neither in P nor NP-complete if $\mathsf{P} \neq \mathsf{NP}$ [Lad75], Otachi and Schweitzer construct a graph class that is neither isomorphism-complete nor isomorphism-tractable, assuming $\mathrm{GI} \notin \mathsf{P}$ [OS13].

Isomorphism-completeness holds for many basic graph classes like bipartite, chordal or regular graphs [cf. BC79], and there are also more advanced results [BO95; BPT96; UTN05; Ueh13].

The isomorphism-tractable classes also have rich literature. For example, efficient isomorphism algorithms have been developed for colored graphs with bounded color class size [Bab79; FHL80], graphs with bounded genus [Mil80; Mil83], bounded degree [Luk82], bounded eigenvalue multiplicity [BGM82], and bounded tree width [Bod90].

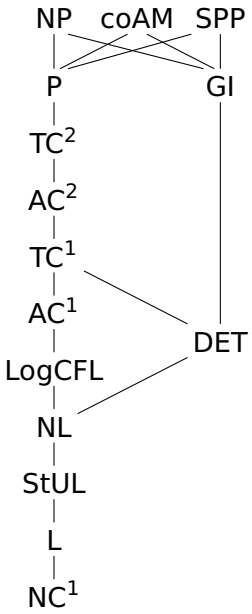

Figure 1.1: Hasse diagramm of the known inclusions between the mentioned complexity classes; GI is the class of all problems that can be reduced to the graph isomorphism problem in polynomial time.

Ponomarenko showed that all graph classes that can be defined by forbidden minors are isomorphism-tractable [Pon88]. Recently, Grohe and Marx generalized this to all graph classes that can be defined by excluding a fixed topological subgraph [GM12]. These graph classes also include graphs of bounded degree, and the algorithm employs Luks' techniques for this case [Luk82] as a subroutine.

For an overview of the status of GI on many different graph classes see also the monograph of Spinrad [Spi03] and the survey of Köbler [Köb06].

The isomorphism-tractable graph classes admit a finer classification through subclasses of P like the NC hierarchy and logspace classes. For several graph classes, the isomorphism problem was first put into P, then parallel NC algorithms were developed, and finally the problem was shown to be in L. This is strong evidence that GI restricted to these graph classes is strictly easier than the general problem, as the latter is DET-hard [Tor04] and therefore NL-hard.

The first such graph class were trees. Edmonds developed the earliest polynomial-time algorithm for tree isomorphism [cf. BS65, Section 6.21]. Linear-time algorithms were independently given by Zemlyachenko [Zem70] and by Hopcroft and Tarjan [HT72; cf. AHU74, Theorem 3.3].[1] In 1991, an AC^1 algorithm was designed by Miller and Reif [MR91], and one year later, Lindell [Lin92] obtained an L upper bound. On the other hand, tree isomorphism is also L-hard [JKM+03],[2] so the known upper and lower complexity bounds for the tree isomorphism problem match.

[1] Edmonds, Scoins, Weinburg, and others found similar algorithms, but did not publish the details [HT72].

[2] This applies when the usual *pointer representation* is used, i.e., graphs are given as lists of edges. When the trees are given in *string representation* (i.e., as nested parenthesis structure), tree isomorphism becomes NC^1-complete [Bus97; JKM+03].

A second precedent, where the complexity of isomorphism testing was successively improved from polynomial time to logspace, is the class of planar graphs. Hopcroft and Tarjan gave a polynomial-time algorithm [HT71]; Hopcroft and Wong improved this to linear time [HW74]. Regarding the parallel complexity, Miller and Reif developed an AC^3 algorithm [MR91] that can be improved to AC^1 when combined with the planar embedding algorithm of Ramachandran and Reif [RR96]. Datta et al. showed that isomorphism of planar graphs can be solved in logspace [DLN+09]; this result can be generalized to graphs that exclude only one of $K_{3,3}$ and K_5 as minor [DNT+09] and to graphs of bounded genus [EK14]. Note that trees are both $K_{3,3}$-free and K_5-free, so the L-hardness mentioned above also applies here.

This thesis contributes to this branch of research by providing logspace isomorphism tests for the classes of k-trees, interval graphs, Helly circular-arc graphs, and proper circular-arc graphs. Variants of these tests also cover concave-round graphs, convex graphs, and circular-convex graphs. An overview of the results is given in Table 1.1 at the end of the introduction.

1.1 Canonization of k-trees

The class of k-trees contains all graphs that can be obtained from the inductive construction that starts with a k-clique and allows to introduce a new vertex if it is connected to all vertices of a previously present k-clique; see Figure 1.2 for an example.

In Chapter 3 it is shown that the isomorphism problem for k-trees is L-complete for each fixed $k \in \mathbb{N}^+$. The first published polynomial-time algorithm for isomorphism of k-trees was obtained by Klawe, Corneil, and Proskurowski [KCP82], who actually cover a larger class of graphs and mention that they have heard of an earlier polynomial-time algorithm for k-trees from Hedetniemi in 1977. The parallel complexity of k-tree isomorphism has been previously investigated by Greco, Sekharan, and Sridhar [GSS02], who showed that it can be solved in AC^2. Grohe and Verbitsky improved this to TC^1 [GV06], also covering a larger class of graphs. One year later, Arvind, Das, and Köbler showed an StUL upper bound for k-tree isomorphism [ADK07].

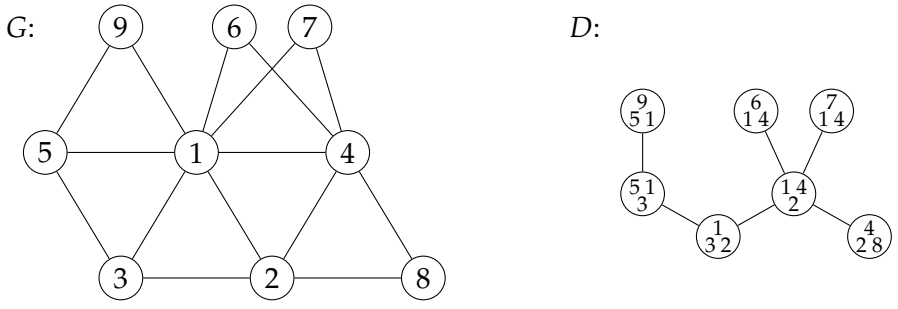

Figure 1.2: A 2-tree G. It can be constructed by starting with the 2-clique $\{1, 2\}$ and then adding the remaining vertices in ascending order, as each of them has exactly two neighbors with a smaller label, which are adjacent. The graph G admits the tree decomposition D of width 2.

In fact, Chapter 3 contains the formally stronger result that canonical labelings for k-trees can be computed in logspace. The *canonization problem* for graphs is to produce a *canonical form* $\text{canon}(G)$ for a given graph G such that $\text{canon}(G)$ is isomorphic to G and $\text{canon}(G_1) = \text{canon}(G_2)$ for any pair of isomorphic graphs G_1 and G_2. Clearly, the isomorphism problem for a class of graphs reduces to computing canonical forms for this class. A *canonical labeling* for G is any isomorphism from G to $\text{canon}(G)$. It is not hard to see that even the search version of GI (i.e., computing an isomorphism between two given graphs in case it exists) as well as the automorphism group problem (i.e., computing a generating set of the automorphism group of a given graph) are both logspace reducible to computing canonical labelings.

To compute canonical labelings for k-trees, the algorithm first transforms the input graph G into an undirected tree $T(G)$ whose nodes are formed by all $(k+1)$-cliques and some k-cliques of G. It then enumerates all valid $(k+1)$-colorings of G; there are exactly $(k+1)!$ of them. Based on such a coloring, it colors the nodes of the tree $T(G)$ to encode additional structural information about G. Finally, a variant of Lindell's algorithm [Lin92] is used to compute a canonical labeling for the colored $T(G)$, from which a canonical labeling for the k-tree G is derived.

The class of k-trees is closely related to graphs of *bounded treewidth*: It is known that a graph has treewidth at most k if and only if it is the subgraph of a k-tree. For this reason, treewidth k graphs are also known as *partial k-trees*. The monograph of Kloks [Klo94] offers a nice introduction to treewidth.

Finding space efficient and parallel algorithms for bounded treewidth graphs is an active research area. Bodlaender gave an $O(n^{k+4.5})$ time isomorphism algorithm [Bod90], and the TC^1 algorithm of Grohe and Verbitsky [GV06] also covers this graph class. Building on the latter result, Köbler and Verbitsky designed a TC^2 algorithm for canonization of bounded treewidth graphs [KV08]. Wagner improved this to AC^1 [Wag11] using techniques of Elberfeld, Jakoby, and Tantau, who showed how to compute a tree decomposition of width k for a given treewidth k graph in logspace [EJT10] (previously, LogCFL was known [Wan94]).

It remains open whether isomorphism of bounded treewidth graphs can be decided in logspace. Das, Torán, and Wagner give a logspace algorithm for isomorphism of tree *distance* width k graphs [DTW12], a subclass of treewidth k graphs that is incomparable to k-trees. They also reduce isomorphism of decomposed bounded treewidth graphs to isomorphism of bounded tree distance width graphs, obtaining a LogCFL algorithm for isomorphism of bounded treewidth graphs [DTW12], which currently is the best known upper bound. The remaining obstacle to put this problem in L is thus the computation of *compatible* tree decompositions in logspace, where compatible means that if two partial k-trees are isomorphic then there should be an isomorphism that maps one decomposition to the other. Unfortunately, the tree decomposition constructed by the logspace algorithm of Elberfeld, Jakoby, and Tantau [EJT10] strongly depends on the names of the vertices, and computing compatible tree decompositions in logspace seems to require new ideas.

Recently, Das, Datta, and Nimbhorkar used the logspace computable tree decomposition presented in this thesis to design logspace algorithms for deciding reachability in directed k-trees and for computing shortest and longest paths in directed acyclic k-trees [DDN13].

The k-tree isomorphism problem, for unbounded k, is isomorphism-complete [KCP82]. Therefore it makes sense to study the fixed parameter tractability of this problem. A problem is called *fixed parameter tractable (FPT)* with respect to some parameter k, if it is solved by an algorithm in time $f(k)\, n^{O(1)}$, where $f(k)$ can be an arbitrary function that does not depend on the input size n. In Section 3.3 it is shown that the k-tree canonization algorithm outlined above can also be implemented in $O\big((k+1)!\, n\big)$ time. This yields the fastest known FPT isomorphism algorithm for k-trees. Previously, Klawe, Corneil, and Proskurowski gave an FPT isomorphism algorithm for general chordal graphs with maximum clique size $k+1$, which runs in $O\big((k+1)!^2\, n^3\big)$ time [KCP82]. Nagoya improved this to $O\big((k+1)!\, n^3\big)$ [Nag01]. Toda gave an isomorphism algorithm that is FPT in the maximum size s of simplicial components, using $O\big((s!\, n)^{O(1)}\big)$ time [Tod06]. While the exact running time of this algorithm is hard to analyze, the parameter s can be smaller than $k+1$ by a factor of up to $\Theta(n)$ and is never larger. For k-trees, however, it always holds that $k \leq s \leq k+1$, because the kernel (as defined in the proof of Theorem 3.4.3) is always a simplicial component. Relatedly, Yamazaki et al. showed how to check isomorphism for graphs of rooted tree distance width k in time $O(k!^2\, k^2\, n^2)$ [YBF+99]. This graph class contains non-chordal graphs and is incomparable to k-trees, but like k-trees it is a subclass of treewidth k graphs. For the isomorphism problem of general treewidth k graphs, Lokshtanov et al. recently gave a $2^{O(k^5 \log k)} n^5$ time algorithm [LPP+14].

Applying the notion of fixed parameter tractability to logspace instead of polynomial time, it is interesting to look for algorithms that take $O\big(f(k) + \log n\big)$ space. The class of problems that admit such algorithms is sometimes called para-L (for parameterized logspace) [EST12]. It can be argued that this is the right notion for 'fixed parameter logspace', as it directly implies an $O\big(f(k) n^{O(1)}\big)$ time bound [see also FG03]. It remains an open question whether isomorphism of k-trees can be solved in para-L: Though the algorithm presented in Chapter 3 can be implemented either in logspace or in FPT time, it is not clear how a $O\big(f(k) + \log n\big)$ space bound can be achieved.

1.2 Interval graphs

An *intersection representation* of a graph G is a mapping α from the vertex set $V(G)$ onto a multiset $\mathcal{M}$ of sets such that vertices u and v of G are adjacent if and only if the sets $\alpha(u)$ and $\alpha(v)$ have a nonempty intersection. Such a multiset $\mathcal{M}$ is an *intersection model* of G. A graph G is an *interval graph* if it admits an intersection model consisting of intervals of reals (or, equivalently, intervals of integers); see Figure 1.3 for an example.

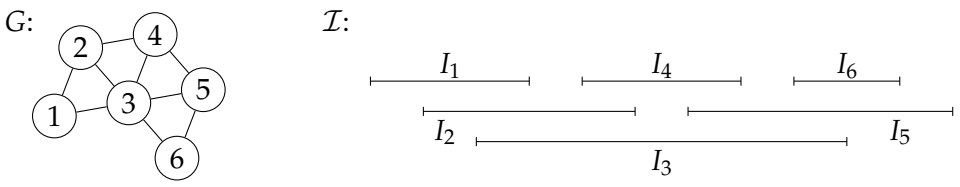

Figure 1.3: An interval graph G and the interval model $\mathcal{I}$ resulting from the interval representation $v \mapsto I_v$.

The algorithmic aspects of interval graphs have been studied persistently for several decades, spurred much by their numerous applications [see e.g. Gol04]. In 1965, Fulkerson and Gross presented the first polynomial-time recognition algorithm for interval graphs [FG65]; it can be implemented in $O(n^4)$ time. Booth and Lueker gave a linear-time algorithm for the same task [BL76], which they followed up with a linear-time algorithm for interval graph isomorphism [LB79]. These algorithms are based on a special data structure called *PQ-tree* that is used to enforce ordering constraints. Korte and Möhring developed a variant of PQ-trees that allows a simpler algorithm [KM89]. By preprocessing the graph's modular decomposition tree, Hsu and Ma [HM99] later presented a simpler linear-time recognition algorithm that avoids the use of PQ-trees altogether. Habib et al. [HMP+00] achieve the same time bound employing the lexicographic breadth-first search of Rose, Tarjan, and Lueker [RTL76] in combination with smart pivoting. Parallel AC^2 recognition and isomorphism algorithms were given by Klein [Kle96].

All of the above algorithms have in common that they compute a *perfect elimination order (PEO)* of the graph's vertices. This ordering has the property that for every vertex, its neighborhood among its successors forms a clique. Fulkerson and Gross [FG65] show that a graph has a PEO if and only if it is chordal.

Recognition of interval graphs in logspace follows from the results of Reif [Rei84] and Reingold [Rei08]. In Chapter 4, a logspace algorithm is described that constructs canonical interval representations: Given an interval graph G, it constructs a function α_G that maps the vertices of G to intervals such that two vertices u and v are adjacent if and only if the intervals $\alpha_G(u)$ and $\alpha_G(v)$ intersect, and isomorphic graphs $G_1 \cong G_2$ are mapped to equal interval models $\alpha_{G_1}(G_1) = \alpha_{G_2}(G_2)$. In particular, this gives another recognition algorithm and moreover implies that testing isomorphism of interval graphs is also possible in logspace. The algorithm uses methods that are optimized for space complexity. As such, it neither relies on computing the graph's PEO nor uses transitive orientation algorithms for comparability graphs as in the approach of Kozen, Vazirani, and Vazirani [KVV85]. Instead, the basis of the algorithm is the observation of Laubner [Lau10] that in an interval graph, the set of maximal cliques and a modular decomposition tree are definable in a certain logical formalism, which makes these objects tractable in logarithmic space.

More specifically, canonical representation of interval graphs is reduced to that of interval *hyper*graphs in Section 4.1. In Section 4.2 it is then shown how these interval hypergraphs can be split into *overlap components* whose interval models are essentially unique and can be computed canonically in logspace using Reingold's algorithm for undirected reachability [Rei08]. These components are then placed in a tree and colored with their canonical interval models. This tree can be canonized using Lindell's algorithm [Lin92]; its canonical form is then used to combine the canonical interval representations of the components to one for the whole hypergraph. The tree used in this algorithm can be viewed as PQ-tree; it encodes all possible interval representations and allows to choose one of the representations in a canonical way. It can be constructed in logspace without the iterative refinement that is inherent to the linear-time algorithms.

A hypergraph is an interval hypergraph if its vertices can be ordered so that each hyperedge consists of vertices that are consecutive in this order. Switching to hypergraphs bears the advantage that these exhibit richer structure; this helps to avoid technical complications and to focus on the essence of the algorithm. Recognition of interval

hypergraphs is clearly equivalent to testing the so-called *consecutive-ones property*: A boolean matrix (which can be viewed as the incidence matrix of a hypergraph) has the consecutive-ones property for rows if its columns can be reordered such that the ones in each row are consecutive. The complexity of testing for this property is similar to that of the recognition of interval graphs: Booth and Lueker gave a linear-time algorithm that uses PQ-trees [BL76], which was later simplified by Hsu and McConnell [HM03]. Parallel AC^2 algorithms were given by Chen and Yesha [CY91] and by Annexstein and Swaminathan [AS98]; an AC^2 algorithm also follows from the parallel algorithms for PQ-trees by Klein and Reif [KR88]. The results in Section 4.2 imply that testing the consecutive-ones property and finding an appropriate column permutation are in logspace (and thus also in AC^1).

Another consequence of the canonical representation algorithm for interval hypergraphs is that convex graphs can be canonized in logspace; this is worked out in Section 4.3. The isomorphism problem for this class was previously known to be decidable in AC^2 [Che96] and in linear time [Che99]. Convex graphs include bipartite permutation graphs. The isomorphism problem for the latter class was only known to be in AC^1 [CY93; YC96]. To the best of the author's knowledge, the logspace algorithm for interval graph isomorphism is the first for a natural class of graphs that contain cliques of arbitrary size. For all graph classes mentioned in this paragraph, the isomorphism problem has a matching lower bound; i.e., it turns out to be logspace complete.

Knowing efficient algorithms for interval graph isomorphism leads to the question whether they can be generalized to larger graph classes. As a graph is interval if and only if it is both chordal and a comparability graph, it is natural to consider these two graph classes. However, Booth and Lueker showed that both chordal graphs and comparability graphs are isomorphism-complete [BL75]. The same is true for *directed path graphs* [BPT96], a graph class between interval graphs and chordal graphs. Another natural generalization are graphs that admit an intersection model that consists of certain geometrical objects on the plane. But even the special case of *grid intersection graphs* is isomorphism-complete [Ueh08], where the geometrical objects are line segments that are parallel to either axis. The survey of Uehara discusses the status of GI on several other graph classes that admit geometrical representations [Ueh14]. Circular-arc graphs, which also generalize interval graphs, are discussed below.

Constrained interval representations

The *interval representation problem* asks for a given graph $G = (V, E)$, if G is an interval graph, and if so, to compute an interval representation α for G, i.e., α determines for each vertex $u \in V$ an interval $\alpha(u)$ such that $E = \left\{\{u, v\} \in \binom{V}{2} \mid \alpha(u) \cap \alpha(v) \neq \emptyset\right\}$. All the interval graph recognition algorithms mentioned above actually solve the interval representation problem, which is thus known to be solvable in linear time [BL76; KM89; HM99; COS09] and in AC^2 [Kle96]. By the results in Chapter 4, it is complete for logspace.

Sometimes interval representations that satisfy additional constraints are desirable. A graph is *proper interval* if it admits an interval representation where no interval contains another. Wegner characterized proper interval graphs by forbidden induced subgraphs [Weg67; cf. DHH96]. In the interval graph G from Figure 1.3, the vertices $\{1, 3, 4, 6\}$ induce one of the forbidden subgraphs, so this graph is not proper interval.

Proper interval representations can be found in linear time by algorithms of Deng, Hell, and Huang [DHH96], Hell, Shamir, and Sharan [HSS01], and Corneil [Cor04]. An AC^2 algorithm is designed by Bang-Jensen, Huang, and Ibarra [BHI07]. In Section 4.4 it is described how canonical proper interval representations can be computed in logspace, implying also logspace recognition of proper interval graphs.

Unit interval graphs are interval graphs representable by systems of intervals that all have the same length. Any such graph is obviously a proper interval graph, and the converse is also true by a classical result of Roberts [Rob69]. Corneil et al. show how to construct a unit interval representation in linear time from a proper one [CKN+95]. In Section 4.4.2 it is shown that this transformation is also possible in logspace.

A generalization of proper interval graphs are the *interval matrices* introduced by McConnell [McC03], which prescribe for each interval in which of the other intervals it should be contained. McConnell showed that interval representations of such matrices can be computed in linear time using so-called Δ trees [McC03], and his techniques can also be adapted to obtain a logspace algorithm [KKV13a]. Section 4.6 describes another logspace algorithm for computing canonical interval representations of interval matrices, which instead relies on the following notion of constrained interval graph representation.

Another way to constrain interval representations α for a graph G is to restrict the lengths of individual intervals and/or pairwise intersections. These restrictions can be given by vertex weights $\ell(v)$ that prescribe the length of each interval $\alpha(v)$ and by edge weights $s(\{u,v\})$ that prescribe the length of each intersection $\alpha(u) \cap \alpha(v)$. An interval representation α of G is *ℓ-respecting* if it satisfies the former conditions, *s-respecting* if it satisfies the latter, and *(ℓ, s)-respecting* if it satisfies both. Fulkerson and Gross show how to find (ℓ, s)-respecting interval representations in $O(n^2)$ time [FG65]. Pe'er and Shamir prove that it is NP-complete to decide if a graph G admits an ℓ-respecting interval representation [PS97]. For the restricted case that the clique order of G is unique, the same authors give a polynomial-time algorithm that can also handle more general constraints on differences between extreme points of intervals. The problem of finding s-respecting interval representations has also been investigated by Yamamoto [Yam07].

In Section 4.5, it is shown how to construct (ℓ, s)-respecting interval representations in linear time or alternatively in logspace, and s-respecting interval representations in $O(nm)$ time or in logspace; here and elsewhere, m denotes the number of edges in the input graph. Since computing ℓ-respecting interval representations is NP-hard, this result illustrates that the information on pairwise intersections is quite helpful.

Klavík, Kratochvíl, and Vyskočil consider a variant of the interval representation problem [KKV11], where additionally to the graph G an interval representation of an induced subgraph of G is given as input. They describe an $O(n^2)$ time algorithm that computes a representation of G (if it exists) which extends the given partial representation.

1.3 Circular-arc graphs

Circular-arc (CA) graphs are graphs that admit an intersection model which consists of arcs on a circle. As every interval model can be viewed as an arc model that leaves part of the circle uncovered, CA graphs are a superclass of interval graphs. The containment is strict, as non-chordal graphs like the one depicted in Figure 1.4 can be CA.

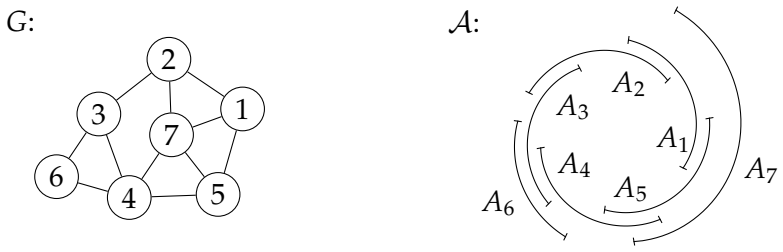

Figure 1.4: A circular-arc graph G and the arc model $\mathcal{A}$ resulting from the arc representation $v \mapsto A_v$.

In his work on interval graphs, Booth conjectured that recognition of CA graphs is NP-complete [Boo75]. This was refuted by Tucker, who gave an $O(n^3)$ time algorithm [Tuc80]. Hsu improved this to $O(nm)$ [Hsu95], Eschen and Spinrad further brought down the runtime to $O(n^2)$ [ES93], and finally McConnell gave a linear time (i.e., $O(n+m)$) algorithm [McC03]. Improving the techniques of Eschen and Spinrad [ES93], Kaplan and Nussbaum obtained a second, much simpler linear-time algorithm [KN11].

Isomorphism of CA graphs remains a challenge up to now: No polynomial-time isomorphism test is currently known, although some approaches have appeared in the literature, of which Uehara gives an overview [Ueh13]. Most widely known is the $O(nm)$ time isomorphism test claimed by Hsu [Hsu95]; only recently a counter-example to its correctness has been given by Curtis et al. [CLM+13].

In quest of efficient recognition and isomorphism algorithms, some natural subclasses of CA graphs besides interval graphs have received special attention. Helly CA graphs, proper CA graphs, and concave-round graphs are the most prominent examples [cf. CLM+13]; see Figure 1.5 for their inclusion structure. In Chapter 5, logspace algorithms are presented that compute canonical representations for these three graph classes.

One building block for these algorithms is a logspace algorithm that computes canonical representations of CA *hyper*graphs; it is described in Section 5.1. This algorithm can also be used to test in logspace whether a given boolean matrix has the *circular-ones property*, that is, whether the columns can be permuted so that the 1-entries in each row form a segment up to a cyclic shift. Note that a matrix has this property if and only if it is the incidence matrix of a CA hypergraph. The recognition problem of the circular-ones property arises in computational biology, namely in analysis of circular genomes [GPZ08; OBS11]. It has similar complexity as recognizing the consecutive-ones property: Prior to the logspace algorithm of Section 5.1, both linear-time algorithms [BL76; HM03] and parallel AC^2 algorithms [CY91; AS98] were known.

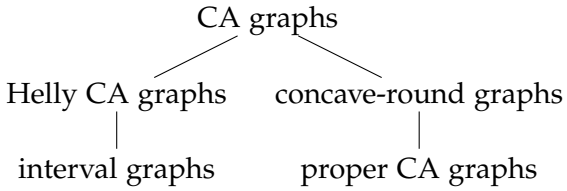

Figure 1.5: Hasse diagram of the inclusions between some classes of circular-arc graphs.

Helly circular-arc graphs

The class of *Helly circular-arc (HCA) graphs* contains all graphs that admit arc models having the *Helly property*, which requires that every subset of arcs with nonempty pairwise intersections has a nonempty overall intersection. For example, the arc model $\mathcal{A}$ in Figure 1.4 satisfies this condition. When computing arc representations for HCA graphs, the resulting arc models are required to be Helly. Since any interval system has the Helly property, the canonical representation problem for HCA graphs generalizes the canonical representation problem for interval graphs. On the other hand, not every arc model is Helly; see Figure 1.6 for examples. Joeris et al. characterize HCA graphs among CA graphs by a family of forbidden induced subgraphs [JLM+11].

HCA graphs were introduced by Gavril under the name *Θ circular-arc graphs*, who gave an $O(n^3)$ time representation algorithm for them [Gav74]. Hsu improved this to $O(nm)$ time [Hsu95], and Joeris et al. obtained an $O(n+m)$ time upper bound [JLM+11]. The fastest known isomorphism algorithm for HCA graphs was developed by Curtis et al. and works also in linear time [CLM+13]. Chen gave a parallel AC^2 algorithm [Che96].

In Section 5.2 it is shown that canonical representations for HCA graphs can be computed in logspace. The first step of the algorithm is to transform a given graph G into a *CA matrix* λ_G that describes the intersection structure in certain arc models of G: For each pair of arcs, it is prescribed whether they should be disjoint, if one should be contained in the other, if they should cover the circle together or if they should overlap otherwise. McConnell observed [McC03] how an CA matrix can be modified to *flip* a subset of arcs in any arc model of it, i.e., to replace each of these arcs with the arc that has the same extreme points but covers the opposite part of the circle. If this flipping operation is applied to a suitable subset of vertices, it vacates a point on the circle and the resulting matrix admits an interval model. The key observation in Section 5.2.2 is that a maximal clique of G can be found in logspace. This clique is then used to flip λ_G into an interval matrix. After computing an interval representation for the latter using the algorithm of Section 4.6, flipping back the arcs of the flipped vertices results in an arc representation of G. Moreover, it is shown in Section 5.2.3 that the resulting arc model is unique up to isomorphism (when some natural restrictions are applied), so the canonical representation algorithm for CA hypergraphs can be used to obtain a canonical arc representation of G.

While the transformation to the CA matrix λ_G is possible for any (also non-Helly) CA graph G, the choice of X crucially relies on the existence of a Helly arc model of G; so this approach does not generalize to arbitrary CA graphs.

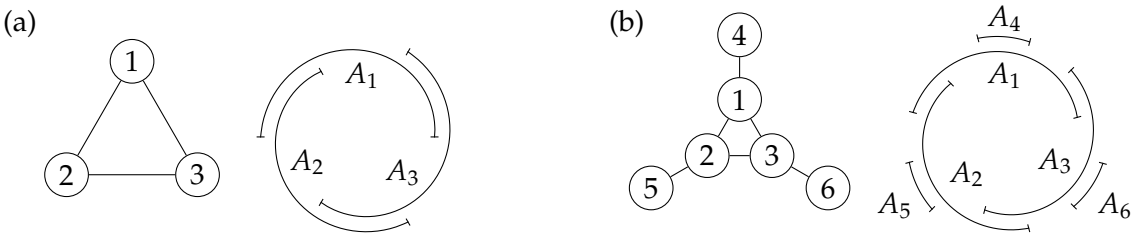

Figure 1.6: Two graphs with non-Helly arc models. The graph in (a) admits an Helly arc model (even an interval model), while the graph in (b) does not.

Note that solvability in logspace implies solvability in AC^1. Previously, no AC^1 algorithm was known for recognition and isomorphism testing of HCA graphs.

It is also interesting that some classes of graphs with Helly intersection models are isomorphism-complete. For example, Uehara shows this for the case of axis-parallel rectangles in the plane [Ueh08].

Proper circular-arc graphs and concave-round graphs

The class of *proper circular-arc (PCA) graphs* consists of all graphs that admit an arc model that is *proper*, i.e., where no arc contains another. In Section 5.3, a logspace algorithm is presented that for a given PCA graph computes canonical representations by proper arc models, where *canonical* again means that isomorphic graphs receive identical models. This algorithm provides a simultaneous solution in logspace of both the recognition and the isomorphism problems for the class of PCA graphs.

Every proper interval graph is PCA, as every proper interval model can also be viewed as a PCA model. Though PCA graphs may thus appear to be close relatives of proper interval graphs, the extension of the result of Section 4.4 achieved here is far from being straightforward. Differences between the two classes of graphs are well known and have led to different algorithmic approaches also in the past [see e.g. DHH96; KN09; LSS08]. One important difference lies in the relationship of these graph classes to interval and CA hypergraphs. Roberts discovered that proper interval graphs admit a natural characterization: A graph G is proper interval if and only if its *neighborhood hypergraph* $\mathcal{N}[G]$ is interval [Rob71], where $\mathcal{N}[G]$ has the same vertex set as G and, for each vertex v of G, has an hyperedge $\mathrm{N}[v]$ that contains v and all its neighbors. The circular-arc world is more complex. While $\mathcal{N}[G]$ is a CA hypergraph whenever G is a PCA graph, the converse is not always true. PCA graphs are properly contained in the class of those graphs whose neighborhood hypergraphs are CA. Graphs with this property are called *concave-round graphs* by Bang-Jensen, Huang, and Yeo [BHY00] and *Γ circular-arc graphs* or *Tucker graphs* by Chen [Che96]. The latter name is justified by Tucker's result [Tuc71] that all these graphs are CA (although not necessarily PCA). Hence, it is natural to consider the problem of constructing arc representations for concave-round graphs. The logspace algorithms of Section 5.3 cover also this case.

The recognition problem for PCA graphs, along with model construction, was solved in linear time by Deng, Hell, and Huang [DHH96] and by Kaplan and Nussbaum [KN09]; and in AC^2 by Chen [Che97]. The isomorphism problem for PCA graphs was solved in linear time by Lin, Soulignac, and Szwarcfiter [LSS08]. In a recent paper [CLM+13], Curtis et al. extend this result to concave-round graphs.

For concave-round graphs, Chen gave AC^2 algorithms for both arc model construction [Che93] and isomorphism [Che96].

Unit CA graphs are CA graphs that admit an arc model where all arcs have equal length. Contrary to what one might expect from the interval setting, where proper interval graphs and unit interval graphs coincide, unit CA graphs are a proper subclass of PCA graphs [Tuc74]. Unit arc representations for such graphs can be computed in linear time [LS08; KN09]. Recently, Soulignac has shown how to do this in logspace [Sou14].

1.4 Published parts

The results for k-trees were obtained jointly with Köbler, and were presented at the *34th International Symposium on Mathematical Foundations of Computer Science (MFCS)* [KK09]. An extended version, which also builds on the earlier work by Arvind, Das, and Köbler [ADK07] and which introduces the FPT algorithm, appeared in *Information and Computation* [ADK+12]. Compared to the latter, the treatment in Chapter 3 uses a simplified coloring of the tree representation, which allows a more efficient implementation and a more elegant analysis.

The algorithms for canonical representation of interval graphs were developed in collaboration with Köbler, Laubner and Verbitsky. The first version of this result was presented at the *37th International Colloquium on Automata, Languages and Programming (ICALP)* [KKL+10] and is also described in Laubner's thesis [Lau11]. In this original version, overlap components are defined not for interval hypergraphs but for interval graphs, and for each of them a partial order of its maximal cliques is computed that allows to find an interval representation. The second version of these results was published in the *SIAM Journal on Computing* [KKL+11]; it adds the algorithm for canonical representation of interval hypergraphs and reduces canonical representation of interval graphs to this problem. Chapter 4 follows this second approach, changing the terminology to be consistent with the later circular-arc results[3] and spelling out the simplified canonical representation algorithm for overlap-connected interval hypergraphs that was first sketched in the survey on interval graph isomorphism published together with Köbler and Verbitsky in the *Computational Complexity Column* of the *Bulletin of the EATCS* [KKV12a]. The algorithms for interval graph representation with given interval and intersection lengths (presented in Section 4.5) were obtained together with Köbler and Watanabe; an extended abstract appeared in the proceedings of the *23rd International Symposium on Algorithms and Computation (ISAAC)* [KKW12].

The results on circular-arc graphs were obtained together with Köbler and Verbitsky. A first algorithm for canonical representation of HCA graphs was presented at the *38th International Symposium Mathematical Foundations of Computer Science (MFCS)* [KKV13a], and a simplified algorithm for the same problem was described in an *arXiv.org* preprint [KKV14]. This thesis uses ideas from both approaches, giving a new proof of the result on interval matrices in Section 4.6 and a more direct argument for the canonicity of the Helly arc representations described in Section 5.2.3. The results for PCA and concave-round graphs were presented at the *32nd Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)* [KKV12b]; an extended version is available on *arXiv.org* [KKV13b].

[3] What is called *interval representation* here was called *interval labeling* there; and what is called *interval model* here was called *interval representation* there. In the case of hypergraphs, there is also a change of meaning: While an interval labeling (old terminology) maps hyperedges to intervals, an interval representation (new terminology) maps vertices to points on the line. This allows more flexible application, e.g. in the proof of Theorem 5.1.1.

Table 1.1: Algorithms for recognition and isomorphism of the mentioned graph classes. If applicable, representation algorithms are also listed.

Class of (hyper)graphs	**Recognition/representation**	**Isomorphism**
k-trees	$O\big(f(k)\,n\big)$ time [KCP82]	$O\big(n^{k+2}\big)$ time [KCP82]
		StUL [ADK07]
	L [by CI88; cf. ADK07]	L Theorem 3.2.4
		$O\big((k+1)!\,n\big)$ time Thm. 3.3.1
interval hypergraphs	LinTime [BL76; Hsu02]	LinTime [by BL76]
	AC^2 [CY91; AS98]	
	L Theorem 4.2.6	L Theorem 4.2.6
interval graphs	LinTime [BL76]	LinTime [LB79]
	AC^2 [Kle96]	AC^2 [Kle96]
	L [by Rei84; Rei08]; Cor. 4.3.1	L Corollary 4.3.1
proper interval graphs	LinTime [DHH96]	LinTime } see interval
	AC^2 [BHI07]	AC^2 } see interval
	L Theorem 4.4.3	L } see interval
CA hypergraphs	LinTime [BL76;Tuc71];[HM03]	LinTime [CLM+13]
	AC^2 [CY91]	
	L Theorem 5.1.1	L Theorem 5.1.1
CA graphs	LinTime [McC03; KN11]	*open* [cf. CLM+13]
HCA graphs	$O(n^3)$ time [Gav74]	LinTime [CLM+13]
	LinTime [JLM+11]	AC^2 [Che96]
	L Theorem 5.2.15	L Theorem 5.2.15
concave-round graphs	LinTime [by BL76; Tuc71]	LinTime [CLM+13]
	AC^2 [Che93]	AC^2 [Che96]
	L Theorem 5.3.6	L Theorem 5.3.6
PCA graphs	LinTime [DHH96; KN09]	LinTime [LSS08]
	AC^2 [Che97]	AC^2 } see concave-round
	L Cor. 5.3.2; Thm. 5.3.6	L } see concave-round

2 Definitions and basic facts

This chapter introduces the concepts and notations that are used in this thesis. For most of them, there are also brief explanations in the glossary starting from page 95.

For a set S, its cardinality is denoted by $|S|$. An order $<$ on S induces the *lexicographic order* on the powerset $\mathcal{P}(S)$, where $A < B$ if the smallest element of the symmetric difference $A \triangle B$ belongs to A. This lexicographic order also generalizes to multisets if the symmetric difference is taken with due regard to multiplicities.

2.1 Graphs

Given a graph G, its vertex set is denoted by $\mathrm{V}(G)$ and its edge set by $\mathrm{E}(G)$. The subgraph induced by a set $U \subseteq \mathrm{V}(G)$ is denoted by $G[U]$, and $G - U$ is a shorthand for $G[\mathrm{V}(G) \setminus U]$. A set $U \subseteq \mathrm{V}(G)$ is *independent* if $\mathrm{E}\big(G[U]\big) = \emptyset$, and a *clique* if $G[U]$ contains all possible edges. A clique is a *maxclique* if it is not contained in a larger clique. The *complement of* G is the graph $\overline{G}$ with $\mathrm{V}(\overline{G}) = \mathrm{V}(G)$ such that two vertices are adjacent in $\overline{G}$ if and only if they are nonadjacent in G.

Given a graph G and two vertices $u, v \in \mathrm{V}(G)$, the *distance* $\mathrm{d}_G(u, v)$ is the length of a shortest path from u to v. The *eccentricity* of a vertex $u \in \mathrm{V}(G)$ is the longest distance to another vertex, i.e., $\mathrm{ecc}_G(u) = \max\big\{\mathrm{d}_G(u, v) \,\big|\, v \in \mathrm{V}(G)\big\}$. The *center* of G consists of all vertices with minimal eccentricity.

The *(open) neighborhood* of a vertex $v \in \mathrm{V}(G)$ is the set $\mathrm{N}_G(v)$ of vertices with distance 1 to v. The *degree* of a vertex $v \in \mathrm{V}(G)$ is $\deg_G(v) = |\mathrm{N}_G(v)|$. A vertex $v \in \mathrm{V}(G)$ is *simplicial* if $\mathrm{N}_G(v)$ is a clique. The set $\mathrm{N}_G[v] = \mathrm{N}_G(v) \cup \{v\}$ is the *closed neighborhood* of v. The common closed neighborhood of two vertices u and v is $\mathrm{N}_G[u, v] = \mathrm{N}_G[u] \cap \mathrm{N}_G[v]$. A vertex u is *universal* if $\mathrm{N}_G[u] = \mathrm{V}(G)$.

Two vertices $u, v \in \mathrm{V}(G)$ with $\mathrm{N}_G(u) = \mathrm{N}_G(v)$ are called *fraternal vertices* (and must be nonadjacent). Similarly, u and v with $\mathrm{N}_G[u] = \mathrm{N}_G[v]$ are called *twins* (and must be adjacent). Note that both these relations are equivalence relations. The *twin class* $[v]$ of a vertex v consists of v itself along with all its twins. Between two different twin classes there are either all possible edges or none. This leads to the notion of the *quotient graph* G' on the vertex set $\mathrm{V}(G') = \big\{[v] \,\big|\, v \in \mathrm{V}(G)\big\}$ where two distinct twin classes $[v]$ and $[u]$ are adjacent if v and u are adjacent in G. The map $v \mapsto [v]$, which is a homomorphism from G to G', will be referred to as the *quotient map*.

2.2 Hypergraphs

A *hypergraph* is a pair $(X, \mathcal{H})$, where X is a set of vertices and $\mathcal{H}$ is a multiset of subsets of X, called *hyperedges*. The same notation $\mathcal{H}$ will be used to denote a hypergraph and its hyperedge set. Similarly to graphs, $\mathrm{V}(\mathcal{H})$ refers to the vertex set X of the hypergraph $\mathcal{H}$.

The *support* of $\mathcal{H}$ is the subset of its vertices that belong to at least one hyperedge and is denoted by $\operatorname{supp}(\mathcal{H}) = \bigcup_{A \in \mathcal{H}} A$. Vertices in $\mathrm{V}(\mathcal{H}) \setminus \operatorname{supp}(\mathcal{H})$ are called *isolated*. A *slot* is an inclusion-maximal subset S of $\mathrm{V}(\mathcal{H})$ such that each hyperedge $A \in \mathcal{H}$ contains either all of S or none of it. *Twins* in a hypergraph are two vertices that are in the same slot.

The *complement* of a hypergraph $\mathcal{H}$ is the hypergraph $\overline{\mathcal{H}} = \{\!\{\overline{A} \mid A \in \mathcal{H}\}\!\}$ on the same vertex set, where $\overline{A} = \mathrm{V}(\mathcal{H}) \setminus A$. Each hyperedge $\overline{A}$ of $\overline{\mathcal{H}}$ inherits the multiplicity of A in $\mathcal{H}$. The *dual* of a hypergraph $\mathcal{H}$ is the hypergraph $\mathcal{H}^{\mathsf{D}} = \{\!\{v^* \mid v \in \mathrm{V}(\mathcal{H})\}\!\}$ on the vertex set $\mathrm{V}(\mathcal{H}^{\mathsf{D}}) = \mathcal{H}$, where $v^* = \{\!\{A \in \mathcal{H} \mid v \in A\}\!\}$. Hyperedges in $\mathcal{H}$ that have multiplicity greater than 1 become twin vertices in $\mathcal{H}^{\mathsf{D}}$ and vice versa.

A hypergraph $\mathcal{H}$ is *proper* if its hyperedges are incomparable by inclusion; in particular, all its hyperedges must have multiplicity 1. A hypergraph $\mathcal{H}$ has the *Helly property* if every subset of pairwise intersecting hyperedges has a common vertex:

$$\forall \mathcal{S} \subseteq \mathcal{H} : \Big(\forall A, B \in \mathcal{S} : A \cap B \neq \emptyset\Big) \Rightarrow \bigcap_{A \in \mathcal{S}} A \neq \emptyset$$

A hypergraph $\mathcal{H}$ is *k-uniform* if all its hyperedges have size k. A graph G can thus be viewed as a 2-uniform hypergraph.

A *(vertex) coloring* of a (hyper)graph $\mathcal{H}$ is a function $c\colon \mathrm{V}(\mathcal{H}) \to C$; the elements of the set C are called *colors*. A *valid coloring with k colors* (or *k-coloring*) of $\mathcal{H}$ is a coloring $c\colon \mathrm{V}(\mathcal{H}) \to C$ with $k = |C|$ that satisfies $c(u) \neq c(v)$ for any two vertices $u \neq v$ that occur together in some hyperedge of $\mathcal{H}$. A *labeling* of a (hyper)graph $\mathcal{H}$ is a bijection $\ell\colon \mathrm{V}(\mathcal{H}) \to \{1, \ldots, |\mathrm{V}(\mathcal{H})|\}$. An *edge-coloring* of a (hyper)graph $\mathcal{H}$ is a function $c\colon \mathcal{H} \to C$.

From graphs to hypergraphs
Given a graph G, its *open neighborhood hypergraph* has the same vertex set as G and its hyperedges are $\mathcal{N}(G) = \{\!\{\mathrm{N}(v) \mid v \in \mathrm{V}(G)\}\!\}$. Similarly, the *(closed) neighborhood hypergraph* of a graph G is defined by $\mathcal{N}[G] = \{\!\{\mathrm{N}[v] \mid v \in \mathrm{V}(G)\}\!\}$.

The *maxclique hypergraph* $\mathcal{C}(G)$ of a graph G has the same vertex set as G and the maxcliques of G as its hyperedges. The *bundle hypergraph* $\mathcal{B}(G)$, which is the dual of $\mathcal{C}(G)$, has the maxcliques of G as vertices and a hyperedge B_v for each vertex v of G, where $\mathrm{B}_v = \{C \in \mathcal{C}(G) \mid v \in C\}$ is the *(maxclique) bundle* of v. Note that for twins $u, v \in \mathrm{V}(G)$, the bundles B_u and B_v are equal; in this case the corresponding hyperedge has multiplicity greater than one.

From hypergraphs to graphs
The *intersection graph* of a hypergraph $\mathcal{H}$ is the graph $\mathbb{I}(\mathcal{H})$ with vertex set $\mathcal{H}$ where A and B are adjacent if and only if they have a nonempty intersection. Note that if $A = B$, these two vertices are twins in the intersection graph.

Two hyperedges $A, B \in \mathcal{H}$ *overlap* (written $A \between B$) if A and B have a nonempty intersection but neither of them includes the other. The *overlap graph* $\mathbb{O}(\mathcal{H})$ is the subgraph of the intersection graph $\mathbb{I}(\mathcal{H})$ where the vertices corresponding to the hyperedges A and B are adjacent if and only if $A \between B$ or $A = B$.

Of course, $\mathbb{O}(\mathcal{H})$ can be disconnected even if $\mathbb{I}(\mathcal{H})$ is connected. A subset $\mathcal{O}$ of the hyperedges of $\mathcal{H}$ corresponding to a connected component of $\mathbb{O}(\mathcal{H})$ will be referred to as an *overlap component* of $\mathcal{H}$. This is a subhypergraph of $\mathcal{H}$ and should not be confused with the corresponding induced subgraph of $\mathbb{O}(\mathcal{H})$. Note that a hyperedge of an overlap component inherits the multiplicity that it has in $\mathcal{H}$.

If $\mathcal{O}$ and $\mathcal{O}'$ are different overlap components, then either every two hyperedges $A \in \mathcal{O}$ and $A' \in \mathcal{O}'$ are disjoint or all hyperedges of one of the two components are contained in a single slot of the other component. Indeed, for $A, B \in \mathcal{O}$ and $A' \in \mathcal{O}'$, the conditions $A \subset A'$, $A \between B$, and $\neg(B \between A')$ imply that $B \subset A'$; and similarly, the conditions $A \cap A' = \emptyset$, $A \between B$ and $\neg(B \between A')$ imply $B \cap A' = \emptyset$. This containment relation between overlap components determines a tree-like decomposition of $\mathcal{H}$.[1] In the case that $\mathbb{O}(\mathcal{H})$ is connected, $\mathcal{H}$ will be called an *overlap-connected hypergraph*.

2.3 Isomorphism

Given two graphs G and H, an *isomorphism* from G to H is a bijection $\varphi\colon \mathrm{V}(G) \to \mathrm{V}(H)$ with $\{u, v\} \in \mathrm{E}(G) \Leftrightarrow \{\varphi(u), \varphi(v)\} \in \mathrm{E}(H)$. This generalizes to hypergraphs: An *isomorphism* from $\mathcal{H}$ to $\mathcal{K}$ is a bijection $\varphi\colon \mathrm{V}(\mathcal{H}) \to \mathrm{V}(\mathcal{K})$ such that every $A \subseteq \mathrm{V}(\mathcal{H})$ is a hyperedge in $\mathcal{H}$ if and only if its image $\varphi(A)$ is a hyperedge in $\mathcal{K}$, and both have the same multiplicity. On colored (hyper)graphs, an isomorphism must additionally preserve colors. G and H are called *isomorphic*, in symbols $G \cong H$, if there is an isomorphism φ from G to H. The image of G under a bijection φ defined on $\mathrm{V}(G)$ is denoted by $\varphi(G)$.

Given a class C of (hyper)graphs, a function f defined on C is an *invariant* for C if

$$\forall G, H \in \mathsf{C} : G \cong H \Rightarrow f(G) = f(H)\,.$$

If the reverse implication also holds, the function f is a *complete invariant* for C. If additionally $f(G)$ is a (hyper)graph isomorphic to G for all $G \in \mathsf{C}$, then f computes *canonical forms* for C. An isomorphism λ_G from G to its canonical form $f(G)$ is called a *canonical labeling*. A function computes canonical labelings for C if it maps each $G \in \mathsf{C}$ to a bijection λ_G defined on $\mathrm{V}(G)$ such that $G \mapsto \lambda_G(G)$ computes canonical forms for C, i.e., $G \cong H$ implies $\lambda_G(G) = \lambda_H(H)$.

The isomorphisms from a (hyper)graph G to itself are called *automorphisms*; they form a group which is denoted by $\mathrm{Aut}(G)$. An automorphism is called *nontrivial* if it is not the identity. The *graph automorphism problem (GA)* is to decide if a graph has a nontrivial automorphism. A (hyper)graph without nontrivial automorphisms is called *rigid*.

2.4 Interval and circular-arc hypergraphs

Consider intervals over the set of positive integers $\mathbb{N}^+$, using the standard notation $[a, b] = \{i \in \mathbb{N}^+ \mid a \le i \le b\}$. Given an interval $I = [a, b]$, its *length* $|I|$ is the number of

[1] The overlap graph, overlap components, and the overlap component tree were introduced by Fulkerson and Gross [FG65].

points $i \in I$.[2] An *interval system* $\mathcal{I}$ is a multiset of intervals. It is always assumed that $\bigcup_{I\in\mathcal{I}} I = [1,k]$ for some k, i.e., shifting and gaps are not allowed. The map $r(x) = k+1-x$ will be called the *mirror reflection*, and the isomorphic interval system $\mathcal{I}^* = r(\mathcal{I})$ will be referred to as the *mirror image* of $\mathcal{I}$. An interval system $\mathcal{I}$ is *mirror-symmetric* if $\mathcal{I}^* = \mathcal{I}$.

A hypergraph $\mathcal{H}$ that is isomorphic to some interval system $\mathcal{I}$ is called *interval hypergraph* [cf. BLS99, Section 8.7]. In this case, $\mathcal{I}$ is an *interval model* of $\mathcal{H}$, and an isomorphism $\rho\colon \mathrm{V}(\mathcal{H}) \to \mathrm{V}(\mathcal{I})$ from $\mathcal{H}$ to $\mathcal{I}$ is an *interval representation* of $\mathcal{H}$.

An interval representation of $\mathcal{H}$ induces a linear order $<_{\mathcal{H}}$ on $\mathrm{V}(\mathcal{H})$ such that each hyperedge $A \in \mathcal{H}$ consists of consecutive points w.r.t. $<_{\mathcal{H}}$. Conversely, any linear order $<$ on $\mathrm{V}(\mathcal{H})$ with this property induces an interval representation $\rho_<$ of $\mathcal{H}$.

A *circular order* on the set $X = \{x_1, \dots, x_n\}$ is a circular successor relation $\prec$, i.e., a directed cycle with the vertices X. In particular, $\mathbb{C}_n$ will denote the n smallest elements of $\mathbb{N}^+$ with the circular order $1 \prec 2 \prec \cdots \prec n \prec 1$. An *arc* $A = [a^-, a^+]$ consists of the points appearing in the directed path from a^- to a^+. The arc $A = \{1, \dots, n\}$ is called *complete*. If $A = [a^-, a^+]$ is not complete, a^- and a^+ are collectively referred to as *extreme points* of A, and individually as the *start point* and the *end point* of A, respectively. The *empty arc* $A = \emptyset$ is also permitted.

An *arc system* $\mathcal{A}$ is a multiset of arcs over some circle $\mathbb{C}_n$. Analogously to the interval case, a hypergraph $\mathcal{H}$ that is isomorphic to some arc system $\mathcal{A}$ is called *CA hypergraph*. In this case, $\mathcal{A}$ is a *arc model* of $\mathcal{H}$, and an isomorphism from $\mathcal{H}$ to $\mathcal{A}$ is an *arc representation* of $\mathcal{H}$. A *CA order* $\prec$ of $\mathcal{H}$ is a circular order of $\mathrm{V}(\mathcal{H})$ such that all hyperedges are arcs w.r.t. $\prec$. Note that an arc representation ρ induces the CA order $\prec_\rho$ defined by $\rho^{-1}(1) \prec_\rho \rho^{-1}(2) \prec_\rho \cdots \prec_\rho \rho^{-1}(n) \prec_\rho \rho^{-1}(1)$. Conversely, a CA order $v_1 \prec v_2 \prec \ldots \prec v_n \prec v_1$ of $\mathcal{H}$ specifies an arc representation $\rho_\prec$ of $\mathcal{H}$ up to rotation.

A function $f\colon \mathcal{H} \mapsto \rho_{\mathcal{H}}$ computes *canonical interval (resp. arc) representations* if $\rho_{\mathcal{H}}$ is an interval (resp. arc) representation for each interval (resp. CA) hypergraph $\mathcal{H}$ and isomorphic input hypergraphs $\mathcal{H} \cong \mathcal{K}$ lead to equal models $\rho_{\mathcal{H}}(\mathcal{H}) = \rho_{\mathcal{K}}(\mathcal{K})$.

An arc system $\mathcal{A}$ is *tight*, if for any two arcs $A = [a^-, a^+]$ and $B = [b^-, b^+]$ with $A \subseteq B$, the difference $B \setminus A$ is also an arc. If neither A nor B is empty or complete, this means that at least one of $a^- = b^-$ and $a^+ = b^+$ must hold. A CA hypergraph is *tight* if it admits a tight arc model; in this case, the corresponding arc representation and CA order are also called *tight*. Recognition of tight CA hypergraphs reduces to recognition of CA hypergraphs. To see this, given a hypergraph $\mathcal{H}$, define its *tightened hypergraph* $\mathcal{H}^{\Subset}$ by $\mathcal{H}^{\Subset} = \mathcal{H} \cup \{A \setminus B \mid A, B \in \mathcal{H}\}$. Then $\mathcal{H}$ is a tight CA hypergraph if and only if $\mathcal{H}^{\Subset}$ is a CA hypergraph (because if $A, B \in \mathcal{H}$ and $\emptyset \neq B \subseteq A$, then for every arc representation ρ of $\mathcal{H}^{\Subset}$ the arc $\rho(B)$ cannot be an inner part of the arc $\rho(A)$). An arc system $\mathcal{A}$ is *sharp* if every point on the circle $\mathrm{V}(\mathcal{A})$ is the extreme point of exactly one arc. An arc system $\mathcal{A}$ is *unit* if all arcs have the same length. As interval systems are a special case of arc systems, the notions *tight*, *sharp* and *unit* also apply to the interval setting.

Given a circular order $\prec$ of a set X, consider the set of all arcs $A \subset X$ w.r.t. $\prec$ except the empty arc $\emptyset$ and the complete arc X. The relation $\prec$ induces a (lexicographic) circular order $\prec^*$ on this set, where $A \prec^* B$ if $a^- = b^-$ and $a^+ \prec b^+$ or if $a^- \prec b^-$, $|A| = |X| - 1$,

[2] This does not coincide with the usual notion of length $|I| = b - a$. However, if I is replaced by the interval $(a - 0.5,\, b + 0.5)$ of reals, then both measures coincide.

and $|B| = 1$. The last two conditions say that A is the longest among all arcs with start point a^- and B is the shortest among all arcs with start point b^-. Let $\mathcal{H}$ be an arc system such that $\emptyset, \mathrm{V}(\mathcal{H}) \notin \mathcal{H}$. Modifying $\prec^*$ by bridging all hyperedges that are not present in $\mathcal{H}$ results in a circular order $\prec_{\mathcal{H}}$ on $\mathcal{H}$: For $A, B \in \mathcal{H}$ define $A \prec_{\mathcal{H}} B$ if either $A \prec^* B$ or if there exist arcs $X_1, \dots, X_k \notin \mathcal{H}$ such that $A \prec^* X_1 \prec^* \dots \prec^* X_k \prec^* B$. The circular order $\prec_{\mathcal{H}}$ on $\mathcal{H}$ will be called *lifted* from the circular order $\prec$ on $\mathrm{V}(\mathcal{H})$.

2.5 Interval and circular-arc graphs

A graph G is an *interval graph* if it is isomorphic to the intersection graph of an interval system $\mathcal{I}$ with $\emptyset \notin \mathcal{I}$. This is equivalent to the standard definition of interval graphs as intersection graphs of real intervals. Indeed, if the intervals in $\mathcal{I}$ are understood as a real intervals, this does not change the intersection graph of $\mathcal{I}$. On the other hand, given a multiset of real intervals $\mathcal{I}'$, denote the set of all extreme points by P. Then the system of discrete intervals induced by $\mathcal{I}'$ on P has the same intersection graph. To obtain an interval system over $\mathbb{N}^+$, it remains to apply the order-preserving bijection from P to $\big[1, |P|\big]$.

Note that interval graphs are not just interval hypergraphs with hyperedges of size 2 (those are exactly unions of disjoint paths, optionally with increased edge multiplicity). This difference stems from the fact that intervals correspond to the vertices of interval graphs and to the hyperedges of interval hypergraphs.

Similar to the interval case, a graph G is a *circular-arc (CA) graph* if it is isomorphic to the intersection graph of an arc system $\mathcal{A}$ with $\emptyset \notin \mathcal{A}$.

An *intersection representation* of a graph G is an isomorphism $\alpha\colon \mathrm{V}(G) \to \mathcal{H}$ from G to the intersection graph $\mathbb{I}(\mathcal{H})$ of a hypergraph $\mathcal{H}$. The hypergraph $\mathcal{H}$ is then called an *intersection model* of G. If $\mathcal{H}$ is an interval system (resp. arc system) with $\emptyset \notin \mathcal{H}$, the function α is called an *interval representation* (resp. *arc representation*) of G, and $\mathcal{H}$ is an *interval model* (resp. *arc model*) of G. If $\mathcal{H}$ is Helly, proper, tight, sharp, or unit, then G and α are also called Helly, proper, tight, sharp, or unit, respectively.

For a graph G and functions $\ell\colon \mathrm{V}(G) \to \mathbb{N}^+$ and $s\colon \mathrm{E}(G) \to \mathbb{N}^+$, an interval representation $\alpha\colon \mathrm{V}(G) \to \mathcal{I}$ of G is called *ℓ-respecting* if $|\alpha(v)| = \ell(v)$ for all $v \in \mathrm{V}(G)$, *s-respecting* if $|\alpha(u) \cap \alpha(v)| = s\big(\{u, v\}\big)$ for all $\{u, v\} \in \mathrm{E}(G)$, and *$(\ell, s)$-respecting* if both conditions hold. An s-respecting interval representation α of G is called *minimal* if there is no s-respecting interval representation α' of G that uses fewer points, i.e., that satisfies $\big|\bigcup_{v \in \mathrm{V}(G)} \alpha'(v)\big| < \big|\bigcup_{v \in \mathrm{V}(G)} \alpha(v)\big|$.

A function $f\colon G \mapsto \alpha_G$ computes *canonical intersection representations* for a class C of intersection graphs if (a) α_G is an intersection representation for each $G \in \mathsf{C}$ and (b) isomorphic input graphs $G \cong H$ lead to equal intersection models $\alpha_G(G) = \alpha_H(H)$. If C is a specific class of intersection graphs and f outputs intersection representations of the appropriate type, then f is said to compute canonical representations of this type. For example, f computes canonical Helly arc representations if C is the class of HCA graphs and f outputs Helly arc representations.

Figure 2.1 shows how an algorithm that computes canonical representations subsumes algorithms for various other problems related to isomorphism and recognition.

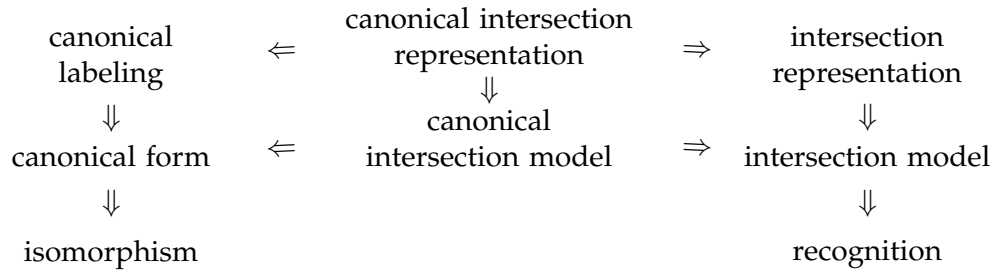

Figure 2.1: Canonical intersection representations allow to solve other problems related to isomorphism and recognition.

2.6 *k*-trees

Fix any $k \in \mathbb{N}^+$. The class of *k-trees* was introduced by Rose [Ros74] and is inductively defined as follows. The complete graph on k vertices is a k-tree. Further, given a k-tree G and a k-clique M in G, one can construct another k-tree by adding a new vertex v and connecting v to every vertex in M. The initial k-clique is called *base* of G, and the k-clique M the new vertex v is connected to is called *support* of v. Note that each k-clique of a k-tree G can be used as base for constructing G – but once the base is fixed, the support of each vertex is uniquely determined. Figure 1.2 on page 3 shows a 2-tree; 1-trees are just trees.

An interesting special case of k-trees are *k-paths*, where the support M_i of any new vertex v_i (except the first vertex added to G) must either contain the vertex v_{i-1} added in the previous step or be equal to the support M_{i-1} of v_{i-1}. Removing vertex 8 from the graph G in Figure 1.2 results in a 2-path. Note that 1-paths are not paths but *caterpillars*, i.e., trees that become paths when all their leaves are removed.

Let G be a graph. A tree T is a *tree decomposition* of G, if each node $M \in \mathrm{V}(T)$ is a subset $M \subseteq \mathrm{V}(G)$ such that

(a) for every edge $\{u, v\} \in \mathrm{E}(G)$, there is a node $M \in \mathrm{V}(T)$ with $\{u, v\} \subseteq M$, and
(b) for every vertex $v \in \mathrm{V}(G)$, the nodes of T that contain v induce a nonempty subtree of T.

The *width* of T is one less than the cardinality of the largest node in $\mathrm{V}(T)$. A graph has *treewidth* k if it admits a tree decomposition of width k, but none of width $k-1$. It is well known that a graph has treewidth at most k if and only if it is a partial k-tree, i.e., a subgraph of a k-tree [see e.g. Klo94].

2.7 Logspace computations

As usual, the class L contains all languages decidable by Turing machines with read-only input tape and an $O(\log N)$ bound on the space used on the working tapes, where N is the input size. The class FL contains all functions computable by Turing machines that additionally have a write-only output tape.

Note that FL is closed under composition: To compute $f\big(g(x)\big)$ for $f, g \in \mathsf{FL}$, simulate the Turing machine for f and keep track of the position of its input head. Every time this

simulation needs a character from f's input tape, simulate the Turing machine for g on input x until it outputs the required character. Note also that g can first output a copy of its input x, so it can be assumed that f has access to both x and $g(x)$. This construction can be iterated a constant number of times, still preserving the logarithmic space bound. This closure property allows to employ pre- and post-processing steps in the description of logspace algorithms.

Logspace algorithms for trees
Many graph problems become solvable in logspace when the input graphs are required to be trees. The following algorithms will help to deal with the tree-like structure that is inherent to many of the problems examined in this thesis.

Fact 2.7.1. *Given a graph G and a node $r \in \mathrm{V}(G)$, in $O(\log n)$ space it can be checked if G is a tree and if so, all edges of G can be directed away from r.*

Proof. Let G' be a directed copy of G where each edge $\{u, v\}$ is replaced by the two arcs (u, v) and (v, u). In a pre-processing step, try the following to compute an Euler tour of G' [cf. AM04, p. 123]: Define a circular order on the neighborhood of each node using the natural order on nodes (given by their names). Start the Euler tour with the lexicographically least arc leaving $w_0 = r$. When reaching some node w_i from w_{i-1}, choose w_{i+1} as the successor of w_{i-1} in the circular order on the neighborhood of w_i. This way, only the two previous nodes have to be remembered. Stop when the arc (w_0, w_1) would be traversed again. If G is a tree, then this results in an Euler tour of G' with the following property: For all nodes v and consecutive occurrences w_i and w_j of v in this tour (i.e., $w_i = w_j = v$ and $w_k \neq v$ for all $k \in \{i+1, \ldots, j-1\}$), it holds that $w_{i+1} = w_{j-1}$. If G contains cycles, this condition will be violated, and if G is not connected, the computed tour will not reach all nodes. This can be checked in $O(\log n)$ space.

To give all edges of G an orientation directed away from r, replace each edge $\{u, v\}$ with the arc (u, v) if the latter precedes (v, u) in the above Euler tour of G'. □

Fact 2.7.2. *Given an undirected tree T and two nodes $u, v \in \mathrm{V}(T)$, the distance $\mathrm{d}_T(u, v)$ can be computed in $O(\log n)$ space.*

Proof. Compute an oriented copy of T with root u using the algorithm of Fact 2.7.1. Then the unique path from v to u can be found by always choosing the unique incoming edge as next step. Only the current node and the number of steps taken so far have to be remembered. Upon reaching u, output the number of steps taken. □

Fact 2.7.3. *The center of an undirected tree T can be computed in $O(\log n)$ space.*

In Lemma 2.8.1 it will be shown that the decision variant of this problem is L-hard even for paths.

Proof. The first step is to show that the eccentricity $\mathrm{ecc}_T(u)$ of each node $u \in \mathrm{V}(T)$ is computable in logspace. This can be done by iterating over all $v \in \mathrm{V}(T)$, each time calculating $\mathrm{d}_T(u, v)$ (this can be done in logspace by Fact 2.7.2). Only the maximum distance has to be remembered, the result being $\mathrm{ecc}_T(u)$.

Observe now that the minimum eccentricity $ecc_{\min}$ of all nodes $u \in \mathrm{V}(T)$ is computable in logspace by iterating over all $u \in \mathrm{V}(T)$. Then compute again the eccentricity of all nodes u, this time outputting u if $\mathrm{ecc}_T(u) = ecc_{\min}$. □

Lindell's logspace algorithm for tree canonization [Lin92] will be used repeatedly throughout this thesis. The following lemma generalizes it to allow vertex colors and to compute not only a canonical form, but a canonical labeling.

Lemma 2.7.4. *Given a colored tree T, which may be rooted or undirected, a canonical labeling of T can be computed in $O(\log n)$ space.*

Proof. Let us first recall Lindell's algorithm for rooted trees and look at the necessary modifications afterwards. Given a rooted tree and an order on the children of each node, this tree can be traversed in logspace: When visiting a node v for the first time, go to its first child, if it has one. If v has no children or if v is visited for the second time, proceed with its next sibling, if it has one. Otherwise, return to its parent. Note that it is only necessary to store the current node and whether we arrived there from its last child.

Lindell's algorithm canonizes a rooted tree by traversing it using a *tree isomorphism order*: Two siblings s_1 and s_2 are ordered $s_1 < s_2$ if the subtree S_1 rooted at s_1 has fewer nodes than the subtree S_2 rooted at s_2, or (for equally large subtrees) if s_1 has fewer children than s_2, or (in case the number of children is also equal) if S_1 has a smaller canonical form than S_2. Lindell shows how to implement the recursion in the latter case without a stack to achieve the overall logspace bound [Lin92].

Colors can be handled by refining the tree isomorphism order with the additional condition $color(s_1) < color(s_2)$ (which gets the highest priority). The canonical labeling can be computed by using a counter i initialized to 0; whenever a node v is visited for the first time in the traversal using the tree isomorphism order, increment i and print '$v \mapsto i$'.

If T is undirected, iterate over the (at most 2) centers c_1 and c_2 of T (which can be found in logspace by Fact 2.7.3), compute a copy T_i of T rooted at c_i using the algorithm of Fact 2.7.1, compute the canonical labeling ψ_i of T_i as before, and choose the canonical labeling ψ_T for T among ψ_1 and ψ_2 so that its image $\psi_T(T)$ becomes minimal. □

2.8 Hardness for logspace

For meaningful L-hardness results, one needs to consider a weak class of reductions. Indeed, every non-trivial problem $P \in \mathsf{L}$ is L-hard under FL reductions, as the reduction can directly decide the input instance and, depending on the result, output a fixed positive or a fixed negative instance of P.

A suitable class of reductions are the first-order translations of Immerman [Imm87], which view instances as logical structures over a vocabulary defined by the problem. A word $x = x_0 \cdots x_{n-1}$ over an alphabet $\Sigma = \{c_1, \ldots, c_k\}$ can be encoded as the structure with the universe $U = [0, n-1]$ and the unary relations $C_j = \{i \in U \mid x_i = c_j\}$ for $1 \leq j \leq k$. When dealing with graphs, it is however more convenient if the edge relation is directly present in the structure. A first-order translation is a many-one reduction, where each relation of the output structure is defined by a first-order formula over the vocabulary of the input structure, the $=$ relation, the successor relation $\prec$, and the constant symbols 0 and *max* for the smallest and largest element w.r.t. $\prec$. To increase the cardinality of the universe, a first-order translation can choose $p \in \mathbb{N}^+$ and encode each variable and constant of the output structure with a p-tuple of variables and constants of the input structure, respectively [cf. Imm87]. First-order translations are equivalent to DLogTime-uniform AC^0 reductions [BIS90].

Etessami showed that the problem

$$\textsc{Ord} = \big\{ \langle P, s, t \rangle \;\big|\; \text{the vertex } s \text{ precedes the vertex } t \text{ on the directed path } P \big\}$$

is L-complete under quantifier-free projections [Ete97], which are a subclass of first-order translations.

The problems

$$\begin{aligned} \textsc{DiPathCenter} &= \big\{ \langle P, c \rangle \;\big|\; \text{the vertex } c \text{ is the center of the directed path } P \big\} \text{ and} \\ \textsc{PathCenter} &= \big\{ \langle P, c \rangle \;\big|\; \text{the vertex } c \text{ is the center of the undirected path } P \big\} \end{aligned}$$

are often useful to prove that the isomorphism and automorphism problems for some graph class are L-hard.

Lemma 2.8.1. DiPathCenter *and* PathCenter *are* L*-complete under first-order translations.*

Proof. Both problems can easily be solved in logspace by traversing the path and counting the number of steps taken.

Regarding the L-hardness, taking the symmetric closure of the edge relation defines a first-order translation from DiPathCenter to PathCenter. Thus it suffices to give a first-order translation from Ord to DiPathCenter.

Let $\langle P, s, t \rangle$ be an instance of Ord, let $b \in \mathrm{V}(P)$ be the vertex without predecessor and let $n \in \mathrm{V}(P)$ be the vertex without successor in P. The first-order translation will construct the DiPathCenter instance $\langle P', c \rangle$, where $c = n$ and

$$\begin{aligned} \mathrm{V}(P') &= V \cup \big\{ i' \;\big|\; i \in V \big\} \cup \big\{ \hat{s} \big\} \\ \mathrm{E}(P') &= \big\{ (i,j) \;\big|\; (i,j) \in E \wedge j \neq t \big\} \cup \big\{ (j',i') \;\big|\; (i,j) \in E \wedge j \notin \{s,t\} \big\} \\ &\quad \cup \big\{ (\hat{s},i') \;\big|\; (i,s) \in E \big\} \cup \big\{ (s',\hat{s}), (t',b), (b',t), (n,n') \big\}. \end{aligned}$$

The path P' thus consists of a forward and a reversed copy of P that are joined together, where the part before the first copy of t is swapped with the part after the second copy of t, and where the second copy of s is duplicated; see Figure 2.2.

If s precedes t in P then n is the center of P' (left side), but if t precedes s then n' is the center of P' (right side).

It remains to show that $\mathrm{V}(P')$, $\mathrm{E}(P')$ and c can be defined using first-order formulas. A vertex u from the first copy of P is encoded as $\langle u, 0 \rangle$, a vertex u' from the second copy of P is encoded as $\langle u, 1 \rangle$ and the second copy $\hat{s}$ of s is encoded as $\langle s, 2 \rangle$. V and E are the vertex and edge relations of P. To improve readability, the shorthands from Table 2.1 are used.

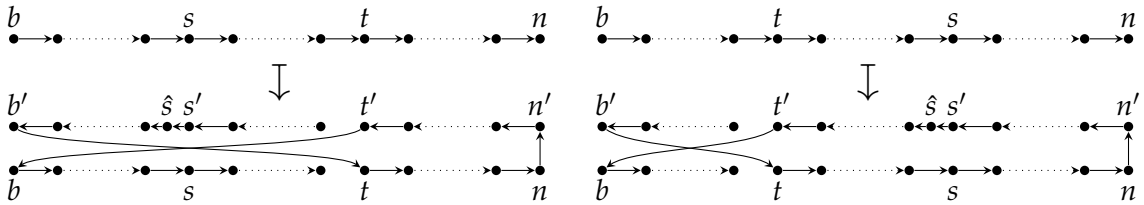

Figure 2.2: Proof of Lemma 2.8.1: The reduction from Ord to DiPathCenter.

Table 2.1: Shorthands for the first-order translation from ORD to DIPATHCENTER.

shorthand	**meaning**
$u = 1$	$0 \prec u$
$u = 2$	$\exists x : 0 \prec x \wedge x \prec u$
$u = b$	$V(u) \wedge \forall x : \neg E(x, u)$
$u = n$	$V(u) \wedge \forall x : \neg E(u, x)$

$$
\begin{aligned}
\varphi_{\mathrm{V}(P')}\big(\langle v_1, v_2\rangle\big) &= \big(V(v_1) \wedge v_2 = 0\big) \vee \big(V(v_1) \wedge v_2 = 1\big) \\
&\quad \vee \big(v_1 = s \wedge v_2 = 2\big) \\
\varphi_{\mathrm{E}(P')}\big(\langle u_1, u_2\rangle, \langle v_1, v_2\rangle\big) &= \big(E(u_1, v_1) \wedge u_2 = 0 \wedge v_2 = 0 \wedge \neg(v_1 = t)\big) \\
&\quad \vee \big(E(v_1, u_1) \wedge u_2 = 1 \wedge v_2 = 1 \wedge \neg(v_1 = s) \wedge \neg(v_1 = t)\big) \\
&\quad \vee \big(u_1 = s \wedge E(v_1, s) \wedge u_2 = 2 \wedge v_2 = 1\big) \\
&\quad \vee \big(u_1 = s \wedge v_1 = s \wedge u_2 = 1 \wedge v_2 = 2\big) \\
&\quad \vee \big(u_1 = t \wedge v_1 = b \wedge u_2 = 1 \wedge v_2 = 0\big) \\
&\quad \vee \big(u_1 = b \wedge v_1 = t \wedge u_2 = 1 \wedge v_2 = 0\big) \\
&\quad \vee \big(u_1 = n \wedge v_1 = n \wedge u_2 = 0 \wedge v_2 = 1\big) \\
\varphi_c\big(\langle v_1, v_2\rangle\big) &= \big(v_1 = n \wedge v_2 = 0\big)
\end{aligned}
$$

□

3 Canonizing k-trees

In this chapter it is shown that isomorphism of k-trees is L-complete for each fixed $k \in \mathbb{N}^+$. In Section 3.1, an algorithm that computes canonical labelings for k-trees is presented and its correctness is proved. Sections 3.2 and 3.3 give logspace and FPT implementations of this algorithm, respectively. In Section 3.4 it is shown that k-tree isomorphism and several simple structural properties of k-trees that can be computed using the tree representation introduced in Section 3.1 are also hard for logspace.

3.1 The algorithm

This section describes the algorithm for k-tree canonization, starting with an overview.

Algorithm 3.1.1. Given a graph G, perform the following steps.

1. Compute an auxiliary graph $T(G)$ (see Definition 3.1.2) and check that G is indeed a k-tree. In this case, $T(G)$ will be called the *tree representation* of G.
2. For each $(k+1)$-coloring c of G (there are exactly $(k+1)!$ by Lemma 3.1.4), perform the following steps:
   a) Compute $T(G, c)$, which is a colored version of the tree representation $T(G)$ (see Definition 3.1.5).
   b) Compute a canonical labeling $\psi_{T(G,c)}$ of $T(G, c)$.
3. Choose a $(k+1)$-coloring c_1 of G such that $\psi_{T(G,c_1)}\big(T(G, c_1)\big)$ becomes minimal.
4. Derive from $\psi_{T(G,c_1)}$ a canonical labeling ψ_G of G (see equation (3.2) on page 28).

The remainder of this section describes this algorithm in more detail and proves that the result is indeed a canonical representation.

The first step is to define the *tree representation* $T(G)$ of a k-tree G. Following the definition, which is illustrated in Figure 3.1, it will be shown that $T(G)$ is a tree decomposition of G. The reasons to choose this particular tree decomposition are that it can be computed efficiently and that $G \cong H$ implies $T(G) \cong T(H)$.

Definition 3.1.2. For a graph G, define $T(G)$ by

$$\mathrm{V}\big(T(G)\big) = \left\{ M \subseteq \mathrm{V}(G) \;\middle|\; \begin{array}{l} M \text{ is a } (k+1)\text{-clique in } G \text{ or } M \text{ is a } k\text{-clique in } G \\ \text{that is not contained in exactly one } (k+1)\text{-clique} \end{array} \right\}$$

$$\mathrm{E}\big(T(G)\big) = \big\{ \{M_1, M_2\} \subseteq \mathrm{V}\big(T(G)\big) \;\big|\; M_1 \subsetneq M_2 \big\}.$$

A useful observation is that k-trees can be characterized in terms of $T(G)$.

Lemma 3.1.3. *G is a k-tree if and only if $T(G)$ is a tree decomposition of G.*

Proof. To prove the 'only if' part, let G be a k-tree, let $\{v_1, \ldots, v_k\}$ be the base k-clique, and let $v_{k+1}, \ldots, v_n$ be the sequence in which the remaining vertices of G were added. Let G_i

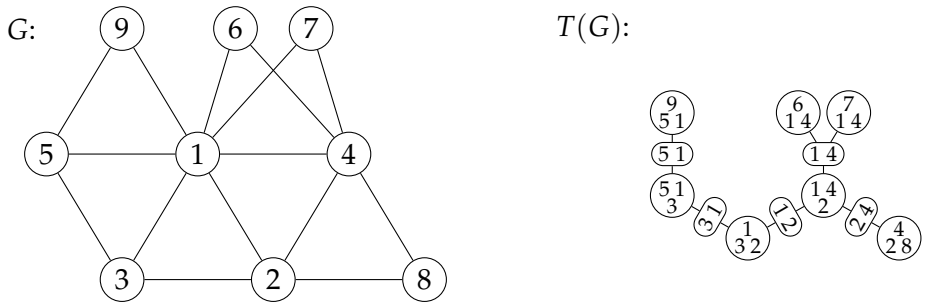

Figure 3.1: The 2-tree G from Figure 1.2 and its tree representation $T(G)$. Compared to the tree decomposition given in Figure 1.2, the tree representation introduces additional nodes to make the tree edges invariant under vertex renaming.

denote $G[\{v_1, \ldots, v_i\}]$. $T(G_k)$ consists of a single k-clique node and $T(G_{k+1})$ consists of a single $(k+1)$-clique node, so they are tree decompositions of G_k and G_{k+1}, respectively. For $i > k+1$, let P_i be the support of v_i, which is a k-clique in G_{i-1}. By inductive hypothesis, $T(G_{i-1})$ is a tree representation of G_{i-1}. If P_i is not a node in $T(G_{i-1})$, it is contained in a unique $(k+1)$-clique node of that tree, and can be added as its neighbor. After that, the $(k+1)$-clique node $M_i = P_i \cup \{v_i\}$ can be added as neighbor of P_i. This results in $T(G_i)$ and can easily be seen to be a tree decomposition of G_i.

For the 'if' part, let G be a graph such that $T(G)$ is a tree decomposition of G. If $T(G)$ consists of a single $(k+1)$-clique node, then G consists only of this clique and thus is a k-tree. Otherwise let M be any k-clique node in $T(G)$ and use it as base of G. All vertices $u \in \mathrm{V}(G) \setminus M$ can be added iteratively: Let M_u be the $(k+1)$-clique that contains u and is closest to M. Then the first k-clique on the path from M_u to M in $T(G)$ can be used as support for u. □

Let us turn to some structural properties of $T(G)$. Note that $T(G)$ has $O(n)$ vertices, so computations involving $T(G)$ can be implemented efficiently.

It will be useful to consider rooted versions of $T(G)$. For $R \in \mathrm{V}\big(T(G)\big)$, let $T_R(G)$ denote the tree representation rooted at R. This allows to identify each $(k+1)$-clique $M \in \mathrm{V}\big(T(G)\big) \setminus \{R\}$ with the unique vertex $v \in M$ that is not present in the parent of M in $T_R(G)$. For later use, denote this vertex by $v_R(M)$ and for each $v \in \mathrm{V}(G) \setminus R$, use $M_R(v)$ to denote the unique $(k+1)$-clique $M \in \mathrm{V}\big(T(G)\big) \setminus \{R\}$ with $v_R(M) = v$. For $v \in R$, let $M_R(v) = R$.

It is clear that the tree representation $T(G)$ might not provide complete structural information about G since it is possible that for an added vertex u, only one of the k edges between u and its support in G can be recovered from $T(G)$. Figure 3.2 shows a 2-tree G' that is not isomorphic to G from Figure 3.1, but has a tree representation $T(G')$ isomorphic to $T(G)$.

The missing information will be encoded using node colors for $T(G)$. Towards this, we will see that every k-tree admits exactly $(k+1)!$ different $(k+1)$-colorings; this is a consequence of the following lemma.

Lemma 3.1.4. *Let G be a k-tree, and let M_0 be a $(k+1)$-clique of G. Any $(k+1)$-coloring of $G[M_0]$ can be uniquely extended to a $(k+1)$-coloring of G.*

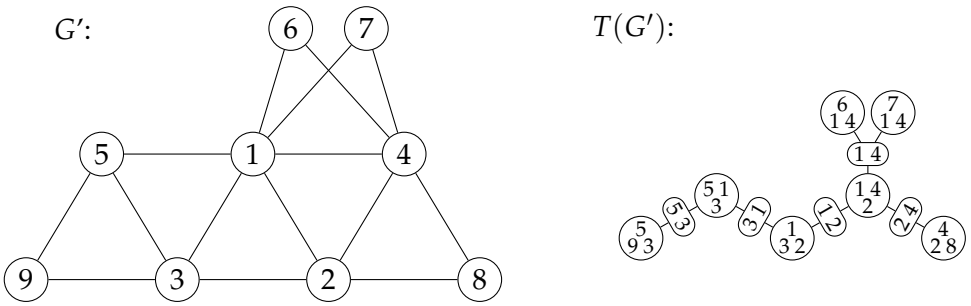

Figure 3.2: A 2-tree G' and its tree representation $T(G')$. G' is not isomorphic to G from Figure 3.1 as it has no node of degree 7, yet $T(G') \cong T(G)$.

Proof. Consider the rooted tree representation $T_{M_0}(G)$. The given coloring fixes the colors of the vertices in the root node. If all vertices in a $(k+1)$-clique M are colored, the same is also true for its children in $T_{M_0}(G)$, as these are k-cliques contained in M. And once all vertices in a k-clique M' are colored, in any child of M' in $T_{M_0}(G)$ only one vertex is not yet colored. As that child is a $(k+1)$-clique, this vertex must receive the unique color not present in M'. □

Definition 3.1.5. Given a k-tree G and a $(k+1)$-coloring c of G, let $T(G,c)$ denote the tree obtained from the tree representation $T(G)$ by coloring every k-clique $M \in \mathrm{V}\big(T(G)\big)$ with the unique color that does not occur in $c(M)$; the $(k+1)$-cliques remain uncolored.

The coloring of $T(G,c)$ was developed by Gainer-Dewar and Gessel [GG14]; it simplifies the original coloring, where each node M of the tree representation was colored with the set $c(M) = \big\{c(v) \bigm| v \in M\big\}$ [ADK+12]. Both these colored trees can be directly constructed from G in logspace, whereas the tree representation used by Arvind, Das, and Köbler [ADK07] (which in turn is related to the decomposition of Klawe, Corneil, and Proskurowski [KCP82]) is defined as the reachable subgraph of a digraph (known as mangrove) derived from G. This allows to decide reachability in the tree $T(G,c)$ in logspace, an essential step to achieve the logspace upper bound in Section 3.2.

The next lemma shows that the colored tree representations of isomorphic $(k+1)$-colored k-trees are also isomorphic.

Lemma 3.1.6. *Let G and H be two k-trees with the $(k+1)$-colorings c and d, respectively, and let φ be a color-preserving isomorphism from (G,c) to (H,d). Then φ, viewed as a mapping from $\mathrm{V}\big(T(G)\big)$ to $\mathrm{V}\big(T(H)\big)$, is a color-preserving isomorphism from $T(G,c)$ to $T(H,d)$.*

Proof. Any isomorphism from G to H maps the k-cliques and $(k+1)$-cliques of G to those of H, preserving inclusions. Thus φ is also an isomorphism from $T(G)$ to $T(H)$. If φ additionally preserves colors, it follows that for each k-clique M of G that $c(M) = d\big(\varphi(M)\big)$, i.e., the same color is missing from M and $\varphi(M)$. Thus φ preserves the colors of the colored tree representations. □

Conversely, the next lemma shows that the isomorphism type of $T(G,c)$ contains complete information about G, proving that the algorithm outlined at the beginning of this section indeed computes a canonical labeling for G.

Lemma 3.1.7. *Let (G, c) be a k-tree with a $(k+1)$-coloring. Then from any colored tree (T', c') that is isomorphic to $T(G, c)$, an isomorphic copy (H, d) of (G, c) can be reconstructed; the result (H, d) only depends on (T', c'). Further, an isomorphism between (G, c) and (H, d) can be constructed from any given isomorphism between $T(G, c)$ and (T', c').*

Proof. For a given colored tree (T', c'), construct (H, d) as follows. Call $t \in \mathrm{V}(T')$ a *$(k+1)$-clique node* if it is uncolored, and a *k-clique node* otherwise. Let $r \in \mathrm{V}(T')$ be the k-clique node with the smallest node name, and consider T' as rooted at r. Let $\mathrm{V}(H) = \{1, \ldots, n\}$, where n is k plus the number of $(k+1)$-clique nodes in T'. Note that G has indeed n vertices due to the one-to-one correspondence between its non-base vertices and its $(k+1)$-cliques. The edges of H and the coloring d will be defined using a mapping M from the nodes of T' to cliques of H, which is defined inductively as follows. For the root node r, let $M(r) = \{1, \ldots, k\}$ and make $M(r)$ a clique in H. Assign these vertices the colors in $\{1, \ldots, k+1\} \setminus c'(r)$ ascendingly, i.e.,

$$\text{for } v \in \{1, \ldots, k\}\colon d(v) = \begin{cases} v & \text{if } v < c'(r) \\ v+1 & \text{otherwise.} \end{cases} \tag{3.1}$$

To extend this to the remaining nodes, let v be the monotone bijection between the $(k+1)$-clique nodes of T' and the remaining vertices $\{k+1, \ldots, n\}$ of G', i.e., $v(t) < v(t') \Leftrightarrow t < t'$, where the latter is the natural order given by the node names. For a $(k+1)$-clique node t of T' with parent p, define $M(t) = M(p) \cup \{v(t)\}$, make $M(t)$ a clique in H, and fix the color $d\big(v(t)\big)$ as the unique color not present in $d\big(M(p)\big)$. For a k-clique node t' of T' with parent p', choose $M(t')$ as the subset of $M(p')$ that drops the vertex colored with $c'(t')$, i.e., $M(t') = \big\{u \in M(p') \;\big|\; d(u) \neq c'(t')\big\}$. This completes the construction of H.

Now let φ be an isomorphism from $T(G, c)$ to (T', c'), let $R = \varphi^{-1}(r)$, and consider $T(G, c)$ to be rooted at R. Construct an isomorphism φ' from (G, c) to (H, d) as follows, where $d|_{M(r)}^{-1}$ is the inverse of d restricted to $M(r) = \{1, \ldots, k\}$, as defined in equation (3.1):

$$\varphi'(u) = \begin{cases} d|_{M(r)}^{-1}\big(c(u)\big) & \text{if } u \in R \\ v\big(\varphi(M_u)\big) & \text{if } u \notin R \end{cases} \tag{3.2}$$

By induction on the depth of M_u in $T(G, c)$, it can be proved that this is indeed a color-preserving isomorphism. □

It remains to show that the canonical labelings of any two isomorphic k-trees G and H map these graphs to the same canonical form $\psi_G(G) = \psi_H(H)$.

Lemma 3.1.8. *Suppose G and H are isomorphic k-trees. Let ψ_G and ψ_H be the labelings computed by the algorithm outlined at the beginning of this section. Then $\psi_G(G) = \psi_H(H)$.*

Proof. Let φ be an isomorphism from G to H, and let M_0 be a $(k+1)$-clique of G. For any $(k+1)$-coloring c of G, its restriction $c|_{M_0}$ to M_0 can be combined with the inverse of φ, resulting in the $(k+1)$-coloring $c|_{M_0} \circ \varphi^{-1}$ of the $(k+1)$-clique $\varphi(M_0)$ of H. By Lemma 3.1.4, a $(k+1)$-coloring of a $(k+1)$-clique uniquely determines a $(k+1)$-coloring of the whole k-tree, so φ is a bijection between the sets $\big\{(G, c) \;\big|\; c \text{ is a } (k+1)\text{-coloring of } G\big\}$ and $\big\{(H, d) \;\big|\; d \text{ is a } (k+1)\text{-coloring of } H\big\}$. As isomorphic colored k-trees lead to isomorphic

colored tree representations by Lemma 3.1.6, the $(k+1)$-colorings c_1 and d_1 that give rise to the lexicographically smallest trees $\psi_{T(G,c_1)}\big(T(G,c_1)\big)$ and $\psi_{T(H,d_1)}\big(T(H,d_1)\big)$ actually make these trees equal. By Lemma 3.1.7, this implies $\psi_G(G) = \psi_H(H)$. □

Note that the algorithm can be extended to colored k-trees as follows. Let $\zeta\colon \mathrm{V}(G) \to C$ be a vertex coloring of G. Modify the coloring of $T(G,c)$ (cf. Definition 3.1.5) as follows: For each $M \in \mathrm{V}\big(T(G)\big)$, add the set $\big\{\zeta(v) \bigm| v \in M\big\}$ to its color.

3.2 Logspace implementation

In this section, it is shown that Algorithm 3.1.1 can be implemented in logspace. The following lemma shows this for step 1.

Lemma 3.2.1. *Given a graph G, the auxiliary graph $T(G)$ can be computed in $O(k \log n)$ space. Also, it can be checked if G is a k-tree within the same space bound.*

Proof. To compute $T(G)$, first iterate over all subsets M of $\mathrm{V}(G)$ of size $k+1$ and output M as a node if M is a $(k+1)$-clique in G. Likewise, find all k-cliques M, and count the vertices $v \in \mathrm{V}(G) \setminus M$ for which $M \cup \{v\}$ is a $(k+1)$-clique. If there is not exactly one such v, then output M as a node, with edges to each such $M \cup \{v\}$. These steps can be implemented using $(k+1)\log n$ space.

To check if G is a k-tree, it suffices by Lemma 3.1.3 to check if $T(G)$ is a tree decomposition of G. So we test if $T(G)$ is a tree. We also test if, for each vertex $v \in \mathrm{V}(G)$, the subgraph of $T(G)$ induced by $\big\{M \in \mathrm{V}\big(T(G)\big) \bigm| v \in M\big\}$ is a tree; by Fact 2.7.1, trees can be recognized in logspace. Finally, we check that all edges $\{u,v\} \in \mathrm{E}(G)$ are contained in some $M \in \mathrm{V}\big(T(G)\big)$, which is clearly possible in logspace. □

Turning to step 2 of the algorithm, let M_0 be a fixed $(k+1)$-clique of G. By Lemma 3.1.4, enumerating all $(k+1)$-colorings of G is essentially the same as enumerating all $(k+1)$-colorings of $G[M_0]$. The following lemma shows that storing the coloring of M_0 is indeed enough.

Lemma 3.2.2. *Given a k-tree G, a $(k+1)$-coloring c of a $(k+1)$-clique of G, and a vertex $v \in \mathrm{V}(G)$, the color of v in the unique extension of c to all of G can be computed in logspace.*

Proof. If $v \in M_0$, its color is already known. Otherwise consider the tree representation $T_{M_0}(G)$ rooted at M_0 (this can be computed in logspace by Fact 2.7.1), and let M_v be the node that contains v and is closest to M_0; it can be found by starting at any node that contains v and following the parent edges as long as v is still contained in the parent node. Following the unique path from M_0 to M_v (which can be done in logspace by following parent edges in $T_{M_v}(G)$), maintain a coloring of the vertices in the current node M: When M is a k-clique, the colors for all vertices in M are already known from the previous node. When M is a $(k+1)$-clique node, assign the vertex which is not in the previous node the unique color that does not occur in the previous node. This is possible in $O(k \log n)$ space. □

Lemma 3.2.3. *For a k-tree G with $(k+1)$-coloring c, the colored tree representation $T(G,c)$ can be computed in $O(k \log n)$ space.*

Proof. $T(G)$ can be computed within this space bound by Lemma 3.2.1. The color of a k-clique node $M \in \mathrm{V}(T(G))$ is the unique color in $\{1,\dots,k+1\} \setminus \{c(v) \mid v \in M\}$ and thus can easily be found in logspace. □

Now we are ready to prove the main result of this chapter.

Theorem 3.2.4. *Given a k-tree G, a canonical labeling ψ_G of G can be computed in $O(k \log n)$ space.*

Proof. The correctness of Algorithm 3.1.1 was shown in Lemma 3.1.8. Lemma 3.2.3 allows to enumerate all $(k+1)$-colorings c of G. The colored tree representation $T(G,c)$ can be obtained in logspace by Lemma 3.2.3. A canonical labeling of $T(G,c)$ can be computed in logspace by Lemma 2.7.4. It remains to observe that the canonical labeling ψ_G for G can be derived as in equation (3.2), which is easily possible in logspace. □

Theorem 3.2.4 immediately yields the following corollaries.

Corollary 3.2.5. *For any fixed k, k-tree (and k-path) isomorphism is* L*-complete.*

The hardness part will be shown in Proposition 3.4.2.

Note that fixing k is essential, as the isomorphism problem for the class of all k-trees, k unbounded, is isomorphism complete [KCP82] and hence unlikely to be decidable in polynomial time.

Furthermore, there is a standard Turing reduction of the automorphism group problem (i.e., computing a generating set of the automorphism group of a given graph) to the search version of GI for colored graphs; a similar reduction exists for counting the number of automorphisms [Hof82; KST93]. It is not hard to see that these reductions can be performed in logspace.

Corollary 3.2.6. *For any fixed k, a generating set of the automorphism group of a given k-tree can be computed in logspace, and hence also a canonical labeling coset for a given k-tree.*

Corollary 3.2.7. *For any fixed k, the number of automorphisms of a given k-tree can be computed in logspace.*

Corollary 3.2.8. *For any fixed k, the k-tree (and k-path) automorphism problem (i.e., deciding whether a given k-tree has a nontrivial automorphism) is* L*-complete.*

Again, the proof of the hardness is postponed; see Proposition 3.4.1.

3.3 Fixed parameter tractability

This section describes a time efficient implementation of the algorithm presented in Section 3.1.

Theorem 3.3.1. *Given a k-tree G, a canonical labeling of G can be computed in $O((k+1)!\, n)$ time.*

Proof. First note that k can easily be obtained from the input graph, as all maxcliques have size $k+1$, and a trivial greedy algorithm finds one of them. The tree representation $T(G)$ of the input graph G can be computed in linear time by iteratively removing simplicial vertices [RTL76]. In this process it can also be checked that G is indeed a k-tree for some k. To enumerate all $(k+1)$-colorings of G, fix any $(k+1)$-clique M_0 of G, enumerate all $(k+1)$-colorings of $G[M_0]$ and extend them in a traversal of $T_{M_0}(G)$ according to the rules given in the proof of Lemma 3.1.4. This immediately yields the colored tree representation $T(G,c)$. Finally, a canonical labeling for $T(G,c)$ can be computed in linear time [AHU74, p. 84]. □

3.4 Complete problems for logspace

This section contains some completeness results for logspace that are related to the algorithms of this chapter. Recall that the problem PathCenter of deciding whether a given vertex c belongs to the center of a given path P is L-complete by Lemma 2.8.1.

Proposition 3.4.1. *For any fixed k, the automorphism problem for k-paths (and thereby k-trees) is* L*-hard under first-order translations.*

Proof. The hardness will be shown via the reduction that maps an instance $\langle P, c\rangle$ of PathCenter to the k-path P', where the neighbors of c are n_1 and n_2, and where

$$\begin{aligned}\mathrm{V}(P') &= \mathrm{V}(P) \cup \{c_1, c_2\}\\ \mathrm{E}(P') &= \big\{\{u,v\} \,\big|\, 1 \le \mathrm{d}_P(u,v) \le k \text{ for } u,v \in \mathrm{V}(P)\big\}\\ &\qquad \cup \big\{\{u,c_i\} \,\big|\, 0 \le \mathrm{d}_{P-\{n_i\}}(u,c) < k \text{ for } u \in \mathrm{V}(P), i \in \{1,2\}\big\}\,.\end{aligned}$$

We may assume that c has distance more than k to both ends, as otherwise the problem is trivial. It is easy to see that P' is a k-path: The first k vertices on the path are the base. As the following vertices are added, the support is always a set of k consecutive nodes on the path, with the additional vertices c_i being added after c enters the support and before c leaves the support, respectively. It is obvious that $P' - \{c_1, c_2\}$ has the nontrivial automorphism that maps the ith vertex on the path to the $(n-i)$th one, but is otherwise rigid. This can be extended to an automorphism of P' (by exchanging c_1 and c_2) if and only if c is the center of P.

It remains to observe that this reduction can be implemented as first-order translation. Indeed, constant bounds on the distance of two vertices can be established using constantly many variables; the condition $\mathrm{d}_P(u,v) \le k$ can for example be expressed as

$$\exists v_2 \cdots \exists v_{k-1} : \big(u = v_2 \vee E(u,v_2)\big) \wedge \big(v_2 = v_3 \vee E(v_2,v_3)\big) \wedge \cdots \wedge \big(v_{k-1} = v \vee E(v_{k-1},v)\big)\,,$$

where E denotes the edge relation of P. □

Proposition 3.4.2. *For any fixed k, the isomorphism problem for k-paths (and thereby k-trees) is* L*-hard under first-order translations.*

Proof. Again, the hardness is shown using a reduction from PathCenter. Modifying the previous construction, the reduction function is $\langle P, c\rangle \mapsto \langle P_1, P_2\rangle$, where the neighbors

of c are n_1 and n_2, the ends of P are v_1 and v_2, and where

$$\begin{aligned}
\mathrm{V}(P_j) &= \mathrm{V}(P) \cup \{c_1, c_2, e\} \\
\mathrm{E}(P_j) &= \big\{\{u,v\} \,\big|\, 1 \leq \mathrm{d}_P(u,v) \leq k \text{ for } u,v \in \mathrm{V}(P)\big\} \\
&\quad \cup \big\{\{u,c_i\} \,\big|\, 0 \leq \mathrm{d}_{P-\{n_i\}}(u,c) < k \text{ for } u \in \mathrm{V}(P), i \in \{1,2\}\big\} \\
&\quad \cup \big\{\{u,e\} \,\big|\, 0 \leq \mathrm{d}_P(u,v_j) < k \text{ for } u \in \mathrm{V}(P)\big\}\,.
\end{aligned}$$

We may assume that c has distance more than $k+1$ to both ends. If c is the center of P, the function that maps the ith vertex of P to the $(n-i)$th, exchanges c_1 and c_2 and maps e_1 and e_2 to themselves is an isomorphism from P_1 to P_2. Conversely, if there is such an isomorphism then the e_i force the original path vertices to be mirrored and the c_i ensure that c is the center. □

Finally, let us examine two problems related to the structure of k-trees. Let G be a graph. Recall that a vertex $v \in \mathrm{V}(G)$ is called simplicial in G if its neighborhood induces a clique. A bijective mapping $\sigma\colon \{1,\ldots,|\mathrm{V}(G)|\} \to \mathrm{V}(G)$ is called *perfect elimination order (PEO)*, if for all i, $\sigma(i)$ is simplicial in $G - \{\sigma(1),\ldots,\sigma(i-1)\}$. Note that a graph can have several perfect elimination orders. It is well known that a graph has a PEO if and only if it is chordal. As k-trees are a subclass of chordal graphs, each k-tree has a PEO.

A related problem is the *fast reordering problem (FRP)* which is employed by Greco, Sekharan, and Sridhar [GSS02] as a preprocessing step for parallel algorithms. It consists of finding a partition $R_0,\ldots,R_\ell$ of $\mathrm{V}(G)$, such that R_ℓ is a clique and each R_i, $i < \ell$, is a maximal independent set of simplicial vertices of $G - \bigcup_{0\leq j<i} R_j$. For general chordal graphs there can be several such sequences, but for k-trees this sequence is unique, and the remaining clique R_ℓ is called the *kernel* of a k-tree G and consists of the vertices in the center node of $T(G)$. Greco, Sekharan, and Sridhar showed that if the input graphs are restricted to k-trees, the FRP can be solved in NC. The following theorem improves this and shows logspace completeness for both problems.

Theorem 3.4.3. *For k-trees (k fixed), it is logspace complete to find a perfect elimination order and to solve the fast reordering problem.*

Proof. Let us first solve the fast reordering problem in logspace. Let G be a k-tree, and let K be the center node of $T(G)$ (as k-clique and $(k+1)$-cliques alternate in any path in G and all leaves are $(k+1)$-cliques, the center is a single node). K is also known as *kernel* of G [GSS02]. For each vertex v of G, denote by

$$l_G(v) = \left\lceil d_{T(G)}(K, M_v)/2 \right\rceil$$

the *level* of v in G, where M_v is the node that contains v and is closest to K (dividing by 2 has the effect of ignoring k-clique nodes for the distance). For each $v \in \mathrm{V}(G)$, its level $l_G(v)$ can be computed in logspace by Lemma 3.2.1 and Fact 2.7.2. Let $l_{\max} = \max\{l_G(v) \,|\, v \in \mathrm{V}(G)\}$. Output $R_i = \{v \in \mathrm{V}(G) \,|\, l_G(v) = l_{\max} - i\}$ for $i = 0,\ldots,l_{\max}$. It follows from the structure of $T(G)$ that $R_0,\ldots,R_{l_{\max}}$ is a solution for FRP.

Next, note that a perfect elimination order can be efficiently computed when a solution $R_0,\ldots,R_\ell$ to the FRP is known (i.e., finding a PEO reduces to solving the FRP): Take the

members of the R_i in ascending order, i.e., first those from R_0, then those from R_1 and so on up to R_ℓ. By the definition of FRP, it follows that the result is a PEO, no matter which order is chosen within each R_i.

The final step is to show that finding a perfect elimination order is hard for logspace even for paths. The result for k-trees (and k-paths) can be obtained by adding a $(k-1)$-clique M and the complete bipartite graph between M and the vertices on the path. An Ord instance $\langle P, s, t\rangle$ can be solved in DLogTime-uniform AC^0 with a single oracle gate for computing a PEO for the path P' defined by

$$\begin{aligned}
\mathrm{V}(P') &= \mathrm{V}(P) \cup \left\{ i' \;\middle|\; i \in \mathrm{V}(P) \setminus \{n\} \right\} \\
\mathrm{E}(P') &= \left\{ \{i, j\} \;\middle|\; (i, j) \in \mathrm{E}(P) \right\} \\
&\quad \cup \left\{ \{i', j'\} \;\middle|\; (i, j) \in \mathrm{E}(P), j \neq n \right\} \cup \left\{ \{i', n\} \;\middle|\; (i, n) \in \mathrm{E}(P) \right\}.
\end{aligned}$$

where n is the vertex in P without successor. Then it holds for any PEO σ of P' (where p_i is a shorthand for the position $\sigma^{-1}(i)$ of a vertex in σ) that

$$\langle P, s, t\rangle \in \textsc{Ord} \Leftrightarrow p_s \leq p_t \leq p_n \vee p_{s'} \leq p_{t'} \leq p_n$$

If $\langle P, s, t\rangle \notin \textsc{Ord}$, then s is between t and n, and s' is between t' and n in P' (right side in Figure 3.3). Thus σ cannot satisfy both $p_s \leq p_t$ and $p_{s'} \leq p_{t'}$. If $\langle P, s, t\rangle \in \textsc{Ord}$ (left side of Figure 3.3), the vertex n does not become simplicial until at least one copy of P is completely removed. Similarly, if the first copy is completely removed before n, the vertex t does not become simplicial before s is removed; and if the second copy is completely removed before n, the vertex t' does not become simplicial before s' is removed. □

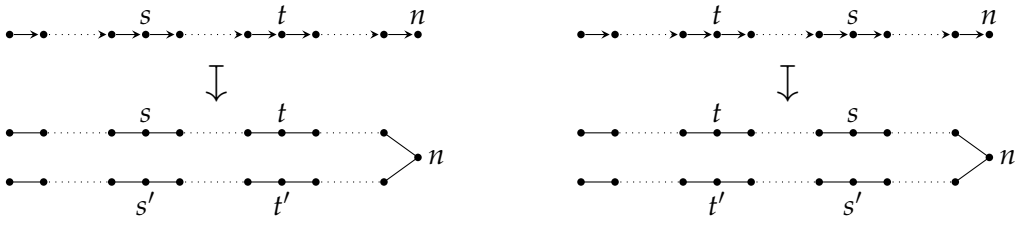

Figure 3.3: Proof of Theorem 3.4.3: The reduction from Ord to finding a PEO.

4 Canonical representation of interval graphs

The main result of this chapter is the logspace algorithm that computes canonical interval representations for interval graphs. Its first step, explained in Section 4.1, is a reduction to canonical representation of interval hypergraphs. The latter problem is solved in Section 4.2: Given an interval hypergraph $\mathcal{H}$, its canonical interval representation $\rho_{\mathcal{H}}$ is computed piecewise, first finding interval representations for each overlap component $\mathcal{O}$ of $\mathcal{H}$. Notice that the overlap components of $\mathcal{H}$ can be computed in logspace using undirected reachability in $\mathbb{O}(\mathcal{H})$ [Rei08]. It turns out that each overlap component $\mathcal{O}$ admits only one interval model $\mathcal{I}$ up to reflection, and this model can be obtained in logspace along with its mirror image $\mathcal{I}^*$. As the canonical version $\mathcal{I}_{\mathcal{O}}$, the algorithm takes the smaller of $\mathcal{I}$ and $\mathcal{I}^*$ (with respect to the lexicographic order on interval systems introduced at the beginning of Chapter 2), or any of them in the mirror-symmetric case $\mathcal{I} = \mathcal{I}^*$. To compose these interval models of the overlap components into a canonical interval model $\mathcal{I}_{\mathcal{H}}$ of the whole hypergraph $\mathcal{H}$, the algorithm uses the tree-like decomposition of $\mathcal{H}$ into overlap components (see Section 2.2) to construct a tree representation $\mathbb{T}(\mathcal{H})$ of $\mathcal{H}$ (see Section 4.2.2). Lindell's tree canonization algorithm [Lin92] will be employed to compute $\mathcal{I}_{\mathcal{H}}$ canonically.

Section 4.3 summarizes the algorithm for canonical representation for interval graphs in logspace. Additionally, it shows how the canonical representation algorithm for interval hypergraphs can be used to compute canonical labelings for convex graphs in logspace.

Turning to constrained interval representations, Section 4.4 describes the algorithms for proper and unit interval representations. Section 4.5 contains the results for representation of interval graphs with prescribed interval and intersection lengths. Section 4.6 gives the logspace algorithm for finding interval representations where the intervals of each pair of adjacent vertices are required to be disjoint, overlapping, contained or containing, respectively.

Complementing the logspace algorithms of this chapter, it is shown in Section 4.7 that the recognition and isomorphism problems of interval and convex graphs are also hard for logspace.

4.1 From interval graphs to interval hypergraphs

Recall that the bundle hypergraph of G is defined by $\mathcal{B}(G) = \{\!\{ \mathrm{B}_v \mid v \in \mathrm{V}(G) \}\!\}$; i.e., it has the maxcliques $\mathcal{C}(G)$ of G as vertices and the maxclique bundles $\mathrm{B}_v = \{ C \in \mathcal{C}(G) \mid v \in C \}$ as hyperedges.

Let us begin with some general properties of the bundle hypergraph that are true for any graph G.

Lemma 4.1.1. *For any graph G, the function $\beta_G \colon V(G) \to \mathcal{B}(G)$ defined by $\beta_G(v) = B_v$ is an intersection representation of G.*

Proof. We have to show that, for any two distinct vertices u and v, $B_u \cap B_v \neq \emptyset$ if and only if u and v are adjacent. Indeed, if $C \in B_u \cap B_v$, then both u and v are in the clique C and hence adjacent. On the other hand, if u and v are adjacent, extend the set $\{u, v\}$ to a maxclique C and notice that $C \in B_u \cap B_v$. Thus, β_G is an intersection representation of G. □

In the course of this section we will see that if G is an interval graph then $\mathcal{B}(G)$ is an interval hypergraph. Given any interval representation ρ of $\mathcal{B}(G)$, Lemma 4.1.1 implies that the function $\rho \circ \beta_G$ is an interval representation of G.

For this approach it is necessary to compute the bundle hypergraph of a given interval graph in logspace. For adjacent vertices u and v in an arbitrary graph G holds: If $N[u, v]$ is a clique, it is maximal. The following lemma shows that, in an interval graph G, any maxclique is of this kind and, hence, can be represented by a pair of vertices u and v (that are adjacent and satisfy the condition that $N[u, v]$ is a clique). An explicit representation of the bundle hypergraph $\mathcal{B}(G)$ of G can be computed in logspace by listing, for each bundle B_v, the maxcliques that contain v.

Lemma 4.1.2. *Every maxclique C of an interval graph G contains vertices u and v such that $C = N[u, v]$.*

Proof. For any $u, v \in C$ it holds that $C \subseteq N[u, v]$; therefore it suffices to find u and v such that $N[u, v] \subseteq C$. For this purpose, consider an interval representation $\alpha \colon V(G) \to \mathcal{I}$ of G and choose $u, v \in C$ so that $\alpha(u) \cap \alpha(v)$ is inclusion-minimal. For each $w \in C$, this implies $\alpha(w) \supseteq \alpha(u) \cap \alpha(v)$ for else $\alpha(u) \cap \alpha(w)$ or $\alpha(w) \cap \alpha(v)$ would be strictly included in $\alpha(u) \cap \alpha(v)$. Suppose now that $z \in N[u, v]$. Since $\alpha(z)$ intersects $\alpha(u) \cap \alpha(v)$, it has nonempty intersection with $\alpha(w)$ for each $w \in C$. By maximality, $z \in C$. □

An interval model $\mathcal{I}$ of an interval graph G is called *minimal* if the size of $V(\mathcal{I})$ is the smallest possible. The following lemma has several important consequences. First, it implies that G is an interval graph if and only if $\mathcal{B}(G)$ is an interval hypergraph [FG65, Theorem 7.1]. Moreover, the bundle hypergraph $\mathcal{B}(G)$ captures all information about a minimal interval model of G, which is unique up to hypergraph isomorphisms. In particular, $\mathcal{B}(G)$ retains the isomorphism type of G, as Lemma 4.1.1 asserts $G \cong \mathbb{I}(\mathcal{B}(G))$. See Figure 4.1 for an example.

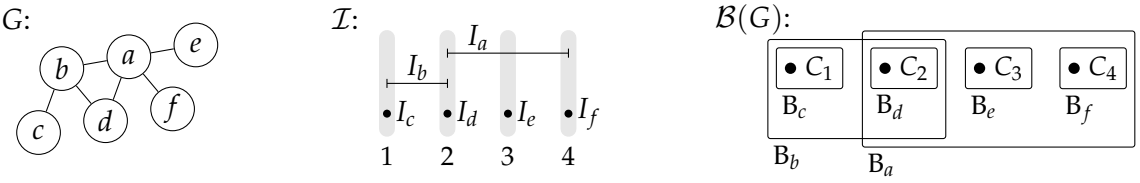

Figure 4.1: An interval graph G, a minimal interval model $\mathcal{I}$ of G, and the bundle hypergraph $\mathcal{B}(G)$ of G. The maxcliques of G are $C_1 = \{b, c\}$, $C_2 = \{a, b, d\}$, $C_3 = \{a, e\}$, and $C_4 = \{a, f\}$.

Lemma 4.1.3. *For every minimal model $\mathcal{I}$ of an interval graph G, the interval system $\mathcal{I}$ viewed as a hypergraph is isomorphic to the bundle hypergraph $\mathcal{B}(G)$ of G.*

Proof. Let $\alpha\colon \mathrm{V}(G) \to \mathcal{I}$ be an interval representation of G such that $\mathcal{I}$ is a minimal interval model. The proof of Lemma 4.1.2 actually shows that every maxclique $C \in \mathcal{C}(G)$ contains vertices u and v such that the three conditions $w \in C$, $\alpha(u) \cap \alpha(v) \subseteq \alpha(w)$, and $\big(\alpha(u) \cap \alpha(v)\big) \cap \alpha(w) \neq \emptyset$ are equivalent. For each maxclique $C = \mathrm{N}[u, v]$, fix a point $x_C \in \alpha(u) \cap \alpha(v)$. The above observation implies

$$w \in C \quad \Leftrightarrow \quad \alpha(w) \ni x_C. \tag{4.1}$$

Note that $x_C \neq x_{C'}$ if $C \neq C'$. Let $X = \big\{x_C \,\big|\, C \in \mathcal{C}(G)\big\}$ and $\alpha'(w) = \alpha(w) \cap X$ for all $w \in \mathrm{V}(G)$. The function α' is still an interval representation of G (with intervals in the linearly ordered set X). Indeed, if u and v are adjacent, let C be a maxclique containing u and v and note that both $\alpha'(u)$ and $\alpha'(v)$ contain x_C; if u and v are nonadjacent, then $\alpha'(u) \cap \alpha'(v) = \emptyset$ because $\alpha(u) \cap \alpha(v) = \emptyset$.

By minimality of $\mathcal{I}$, it follows that $\alpha' = \alpha$ and $\mathrm{V}(\mathcal{I}) = X$. Therefore, the correspondence $C \mapsto x_C$ is a bijection from $\mathcal{C}(G)$ to $\mathrm{V}(\mathcal{I})$. By equation (4.1), this is actually a hypergraph isomorphism from $\mathcal{B}(G)$ to $\mathcal{I}$ (since the condition $w \in C$ can be rewritten as $C \in \mathrm{B}_w$). □

While it is often assumed that the extreme points of the intervals of an interval system are distinct, Lemma 4.1.3 motivates the focus on computing minimal interval representations, where some of the extreme points coincide. This way, arbitrary choices that are not inherent to the canonical representation problem can be avoided. It is easy to see that the two notions are equivalent: To obtain a minimal interval model from an interval model with distinct extreme points, contract each block of consecutive start points together with the following block of consecutive end points to a single point. For the converse direction, replace each point by a set of consecutive points, one for each interval that starts or ends at this point, placing the start points first. If the start points are ordered by interval length and the end points by the corresponding start points, this transformation preserves canonicity.

Recall that a function $G \mapsto \alpha_G$ computes canonical interval representations for interval graphs if (a) α_G is an interval representation of G and (b) $\alpha_G(G) = \alpha_H(H)$ whenever $G \cong H$. Similarly, a function $\mathcal{H} \mapsto \rho_{\mathcal{H}}$ computes canonical interval representations for interval hypergraphs if (a) $\rho_{\mathcal{H}}$ is an interval representation of $\mathcal{H}$ and (b) $\rho_{\mathcal{H}}(\mathcal{H}) = \rho_{\mathcal{K}}(\mathcal{K})$ whenever $\mathcal{H} \cong \mathcal{K}$.

Lemma 4.1.4. *Computing canonical interval representations for interval graphs is reducible in logspace to computing canonical interval representations for interval hypergraphs.*

Proof. Let G be an interval graph. By Lemma 4.1.3, its bundle hypergraph $\mathcal{H} = \mathcal{B}(G)$ is interval. Define an interval representation α_G for G using the canonical interval representation $\rho_{\mathcal{H}}$ of $\mathcal{H}$ by setting $\alpha_G(v) = \rho_{\mathcal{H}}(\mathrm{B}_v) = \big\{\rho_{\mathcal{H}}(C) \,\big|\, C \in \mathrm{B}_v\big\}$ for each vertex v of G. Since the correspondence $v \mapsto \mathrm{B}_v$ is a graph isomorphism from G to $\mathbb{I}(\mathcal{H})$ by Lemma 4.1.1 and because $\rho_{\mathcal{H}}$ is a hypergraph isomorphism from $\mathcal{H}$ to the interval system $\rho_{\mathcal{H}}(\mathcal{H})$, the map α_G is an isomorphism from G to $\mathbb{I}\big(\rho_{\mathcal{H}}(\mathcal{H})\big)$. This shows that α_G is indeed an interval representation of G. It is canonical because $G \cong G'$ implies $\mathcal{B}(G) \cong \mathcal{B}(G')$ and because $\rho_{\mathcal{H}}$ is canonical. It remains to note that the bundle hypergraph $\mathcal{B}(G)$ and the map $v \mapsto \mathrm{B}_v$ are constructible in logspace because of Lemma 4.1.2. □

4.2 Canonical representation of interval hypergraphs

In this section it is shown how canonical representations of interval hypergraphs can be computed in logspace.

Let $\mathcal{H}$ be an interval hypergraph. We may assume that $\mathcal{H}$ is connected, as adding an additional hyperedge $B_0 = \mathrm{V}(\mathcal{H})$ does not change what is an interval representation of $\mathcal{H}$.

Let $\rho\colon \mathrm{V}(\mathcal{H}) \to \mathrm{V}(\mathcal{I})$ be an interval representation of $\mathcal{H}$. Note that ρ takes each slot of $\mathcal{H}$ onto a slot of $\mathcal{I}$ and that the slots of $\mathcal{I}$ form a partition of $\mathrm{supp}(\mathcal{I})$ into intervals. Thus, ρ determines a natural geometric order $<_\rho$ between the slots of $\mathcal{H}$: For two slots S and S', let $S <_\rho S'$ if $\rho(S)$ lies completely to the left of $\rho(S')$ in $\mathbb{N}^+$. A slot order $<$ of $\mathcal{H}$ is called *permissible*, if it does not split any hyperedge $A \in \mathcal{H}$, i.e., if there are no three slots $S_1 < S_2 < S_3$ with $S_1, S_3 \subseteq A \not\supseteq S_2$. For each interval representation ρ, the corresponding slot order $<_\rho$ is permissible. Conversely, any permissible slot order $<$ of $\mathcal{H}$ specifies an interval representation of $\mathcal{H}$ up to permutation of twins (i.e., vertices in the same slot). As both these transformations are easily possible in logspace and as permutations of twins do not change the resulting interval model, the two notions are equivalent for the purposes of this section.

4.2.1 Canonical representations for overlap components

Lemma 4.2.1. *Let $\mathcal{H}$ be an overlap-connected hypergraph with more than one slot and without isolated vertices. If $\mathcal{H}$ is an interval hypergraph, it admits exactly two permissible slot orders, which are the reversal of each other. Moreover, there is a logspace algorithm that computes these slot orders $<$ and $<^*$ or detects that none exist. In the positive case, the algorithm also computes the interval models $\mathcal{I}$ and $\mathcal{I}^*$ defined by $<$ and $<^*$.*

The uniqueness part of Lemma 4.2.1 can be deduced from an equivalent statement by Chen and Yesha [CY91, Theorem 2].

Proof. Let $A_1, \ldots, A_N$ be a spanning walk of the overlap graph $\mathbb{O}(\mathcal{H})$, i.e., $A_k \between A_{k+1}$ for $1 \leq k \leq N-1$ and each $A \in \mathcal{H}$ occurs at least once. Such a walk can be computed in logspace using Reingold's universal exploration sequences [Rei08]. Iterating over the hyperedges A_k in this walk, compute an interval $I_k = [l_k, r_k]$ for each A_k. Once the first interval $I_1 = [1, |A_1|]$ is fixed, the cardinalities of A_2 and $A_1 \cap A_2$ leave only two possibilities for I_2 (resulting in reflected models), and once I_{k-2} and I_{k-1} are fixed, I_k is uniquely determined; see Figure 4.2. As only the two previous intervals have to be remembered, this computation is possible in logspace.

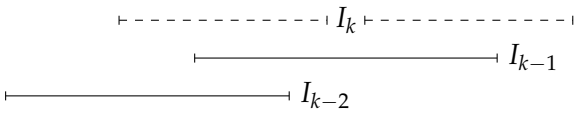

Figure 4.2: Proof of Lemma 4.2.1: Let $A_k \between A_{k-1} \between A_{k-2}$. If I_{k-1} is already determined, only two positions for I_k satisfy $|I_k| = |A_k|$ and $|I_{k-1} \cap I_k| = |A_{k-1} \cap A_k|$. If also I_{k-2} is given, at most one of them satisfies $|I_{k-2} \cap I_k| = |A_{k-2} \cap A_k|$.

If $\mathcal{H}$ is an overlap-connected interval hypergraph, this procedure results in consistent intervals (i.e., $I_j = I_k$ whenever $A_j = A_k$) and the computed intervals moreover satisfy $|I_j \cap I_k| = |A_j \cap A_k|$ for all $j,k \in \{1,\dots,N\}$. If either of these conditions is violated, the algorithm rejects the input hypergraph as non-interval. Otherwise, the algorithm computes the permissible slot order $<$ induced by $A_k \mapsto I_k$, i.e., it defines $S < S'$ by $p(S) < p(S')$, where $p(S) = \left\{x \in \cup_k I_k \;\middle|\; x \in I_k \Leftrightarrow S \subseteq A_k\right\}$ is the set of points where the slot S ends up at. The algorithm returns the slot order $<$ along with its reversal $<^* = <^{-1}$.

To compute the interval model $\mathcal{I}$ corresponding to $<$, the algorithm finds the smallest used point $c = \min \bigcup_k I_k$ and lets $\mathcal{I} = \left\{[l_k - c + 1, r_k - c + 1] \;\middle|\; A \in \mathcal{H}, A = A_k\right\}$. The interval model $\mathcal{I}^*$ corresponding to $<^*$ can easily be obtained in logspace as the mirror-reflection of $\mathcal{I}$. □

If $\mathcal{I}$ and $\mathcal{I}^*$ are equal, the overlap-connected interval hypergraph $\mathcal{H}$ is called *mirror-symmetric* and its interval model is unique. Otherwise, the lexicographically smaller of $\mathcal{I}$ and $\mathcal{I}^*$ can be chosen as the canonical model of $\mathcal{H}$.

4.2.2 The tree representation

As noted before, the overlap components of $\mathcal{H}$ form a tree. Specifically, let S be a slot of an overlap component $\mathcal{O}$ of $\mathcal{H}$. An overlap component $\mathcal{Q}$ of $\mathcal{H}$ is *located at* slot S of $\mathcal{O}$ if $\operatorname{supp}(\mathcal{Q}) \subseteq S$ and there is no 'intermediate' overlap component $\mathcal{O}' \neq \mathcal{O}$ such that $\operatorname{supp}(\mathcal{O}') \subseteq S$ and $\mathcal{Q}$ is contained in some slot of $\mathcal{O}'$. Furthermore, a vertex v of $\mathcal{H}$ is *located at* slot S of $\mathcal{O}$ if $v \in S$ and there is no overlap component $\mathcal{O}'$ located at slot S of $\mathcal{O}$ such that $v \in \operatorname{supp}(\mathcal{O}')$.

The *overlap component tree* of $\mathcal{H}$ is defined as follows: Its nodes are the overlap components of $\mathcal{H}$, their slots, and the vertices of $\mathcal{H}$. Since a slot S of $\mathcal{O}$ may belong also to another overlap component, the corresponding slot node is denoted by $S_\mathcal{O}$. The children of an overlap component node $\mathcal{O}$ are the slots of $\mathcal{O}$. The children of a slot node $S_\mathcal{O}$ are the vertices and the overlap components located at the slot S of $\mathcal{O}$. As $\mathcal{H}$ is connected, there is an overlap component $\mathcal{O}_0$ with $\operatorname{supp}(\mathcal{H}) = \operatorname{supp}(\mathcal{O}_0)$. Thus, $\mathcal{O}_0$ is the root of the overlap component tree. See Figure 4.3 for an example.

Note that the overlap component tree of $\mathcal{H}$ is essentially a *PQ-tree* that encodes all possible interval representations of $\mathcal{H}$ [cf. BL76]. Start with any interval representation of $\mathcal{H}$, and order sibling nodes in the tree by their position in this representation. Slot nodes are *P-nodes*; i.e., their children can be reordered arbitrarily. Overlap component nodes are *Q-nodes*; i.e., the order of their children can only be reversed. It is easy to verify that, after reordering in this way, reading the vertex nodes from left to right again corresponds to an interval representation of $\mathcal{H}$.

Using Lemma 4.2.1 and a simple inductive argument on the depth of the overlap component tree, one can easily show that every interval representation of $\mathcal{H}$ can be obtained in this way [cf. HM03]. It may be of independent interest to note that Lemma 4.2.1 implies logspace constructibility of a PQ-tree for $\mathcal{H}$ and, using this, logspace constructibility of a (non-canonical) interval representation for $\mathcal{H}$; however, this fact is not used below.

In order to obtain a canonical interval representation of an interval hypergraph $\mathcal{H}$, a tree representation $\mathbb{T}(\mathcal{H})$ will be defined in such a way that $\mathcal{H}_1 \cong \mathcal{H}_2$ if and only if $\mathbb{T}(\mathcal{H}_1) \cong \mathbb{T}(\mathcal{H}_2)$. To achieve this, let us start with the overlap component tree of $\mathcal{H}$ and

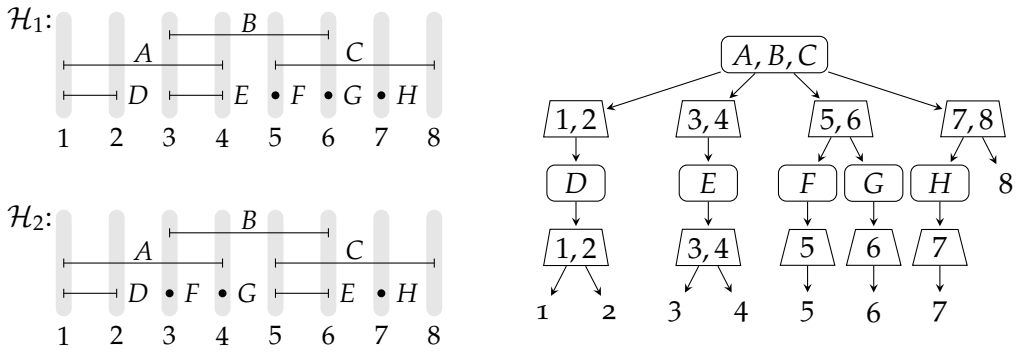

Figure 4.3: An interval hypergraph $\mathcal{H}_1$ and its overlap component tree. In the tree, the node of an overlap component $\mathcal{O}$ is given by listing the hyperedges in $\mathcal{O}$; a slot node $S_\mathcal{O}$ is given by listing the vertices contained in S (the reference to $\mathcal{O}$ is omitted as it is understood from the tree structure). The hypergraph $\mathcal{H}_2$ is not isomorphic to $\mathcal{H}_1$, yet both have isomorphic overlap component trees.

color the component nodes with their canonical interval model. However, this is not enough to ensure that isomorphic tree representations imply isomorphic hypergraphs, because the tree isomorphisms are not restricted in the way they permute the children of an overlap component node. An example where this is a problem can be seen in Figure 4.3.

To make sure that isomorphic trees imply isomorphic hypergraphs, it is necessary to constrain the order of the slot nodes of an overlap component $\mathcal{O}$. Let $<_{\mathcal{O},1}$ and $<_{\mathcal{O},2}$ be the two slot orders of $\mathcal{O}$ provided by Lemma 4.2.1, and let $\mathcal{I}_{\mathcal{O},1}$ and $\mathcal{I}_{\mathcal{O},2}$ be the corresponding interval models. We may suppose w.l.o.g. that $\mathcal{I}_{\mathcal{O},1} \leq \mathcal{I}_{\mathcal{O},2}$. For a slot S of $\mathcal{O}$ and $i \in \{1,2\}$, define the position of S w.r.t. $<_{\mathcal{O},i}$ by

$$\mathrm{pos}_{\mathcal{O},i}(S) = \big|\{S' \mid S' \text{ is a slot of } \mathcal{O} \text{ with } S' \leq_{\mathcal{O},i} S\}\big|.$$

For each overlap component $\mathcal{O}$ that is not mirror-symmetric (i.e., $\mathcal{I}_{\mathcal{O},1} < \mathcal{I}_{\mathcal{O},2}$), the order of its slot nodes can be completely fixed by coloring the slot node $S_\mathcal{O}$ with the color $\mathrm{pos}_{\mathcal{O},1}(S)$.

For each mirror-symmetric overlap component $\mathcal{O}$, the order of its slot nodes $S_\mathcal{O}$ must be fixed up to reflection. To this end, each $S_\mathcal{O}$ will be colored with the minimum of $\mathrm{pos}_{\mathcal{O},1}(S)$ and $\mathrm{pos}_{\mathcal{O},2}(S)$. After this, any two slot nodes with the same color remain interchangeable, and the problem shown in Figure 4.3 persists. To overcome this, the slots of the overlap components will be grouped by introducing an additional type of node in the tree: For each overlap component $\mathcal{O}$, add three nodes $\mathrm{lo}_\mathcal{O}$, $\mathrm{mi}_\mathcal{O}$, and $\mathrm{hi}_\mathcal{O}$ in a layer between $\mathcal{O}$ and its slots. If $\mathcal{O}$ has a mirror-symmetric canonical model $\mathcal{I}_{\mathcal{O},1} = \mathcal{I}_{\mathcal{O},2}$, call a slot S of $\mathcal{O}$ *low* if $\mathrm{pos}_{\mathcal{O},1}(S) < \mathrm{pos}_{\mathcal{O},2}(S)$, *middle* if $\mathrm{pos}_{\mathcal{O},1}(S) = \mathrm{pos}_{\mathcal{O},2}(S)$, and *high* if $\mathrm{pos}_{\mathcal{O},1}(S) > \mathrm{pos}_{\mathcal{O},2}(S)$. If the canonical model of $\mathcal{O}$ is not mirror-symmetric, call all its slots *low*. The low, middle, and high slots of $\mathcal{O}$ become children of the nodes $\mathrm{lo}_\mathcal{O}$, $\mathrm{mi}_\mathcal{O}$, and $\mathrm{hi}_\mathcal{O}$, respectively. Note that, if $\mathcal{O}$ is mirror-symmetric, the node names $\mathrm{lo}_\mathcal{O}$ and $\mathrm{hi}_\mathcal{O}$ are interchangeable.

Using these notions, the tree representation of an interval hypergraph can now be defined formally.

Definition 4.2.2. For a connected interval hypergraph $\mathcal{H}$, its *tree representation* $\mathbb{T}(\mathcal{H})$ is defined by

$$\begin{aligned}
\mathrm{V}(\mathbb{T}(\mathcal{H})) &= \{\mathcal{O}, \mathrm{lo}_\mathcal{O}, \mathrm{mi}_\mathcal{O}, \mathrm{hi}_\mathcal{O} \mid \mathcal{O} \text{ is an overlap component of } \mathcal{H}\} \\
&\quad \cup \{S_\mathcal{O} \mid S \text{ is a slot of an overlap component } \mathcal{O}\} \cup \mathrm{V}(\mathcal{H}), \\
\mathrm{E}(\mathbb{T}(\mathcal{H})) &= \{(\mathcal{O}, \mathrm{lo}_\mathcal{O}), (\mathcal{O}, \mathrm{mi}_\mathcal{O}), (\mathcal{O}, \mathrm{hi}_\mathcal{O}) \mid \mathcal{O} \text{ is an overlap component of } \mathcal{H}\} \\
&\quad \cup \{(\mathrm{lo}_\mathcal{O}, S_\mathcal{O}), (\mathrm{mi}_\mathcal{O}, S_\mathcal{O}), (\mathrm{hi}_\mathcal{O}, S_\mathcal{O}) \mid S \text{ is a low/middle/high slot of } \mathcal{O}\} \\
&\quad \cup \{(S_\mathcal{O}, \mathcal{O}') \mid \text{the overlap component } \mathcal{O}' \text{ is located at slot } S \text{ of } \mathcal{O}\} \\
&\quad \cup \{(S_\mathcal{O}, v) \mid \text{the vertex } v \text{ is located at slot } S \text{ of overlap component } \mathcal{O}\}.
\end{aligned}$$

Furthermore, define a coloring c of the component and the slot nodes by

$$c(\mathcal{O}) = \mathcal{I}_{\mathcal{O},1},$$
$$c(S_\mathcal{O}) = \begin{cases} \mathrm{pos}_{\mathcal{O},1}(S) & \text{if } S \text{ is a low or middle slot of } \mathcal{O}, \\ \mathrm{pos}_{\mathcal{O},2}(S) & \text{if } S \text{ is a high slot of } \mathcal{O}. \end{cases}$$

See Figure 4.4 for an example of a tree representation.

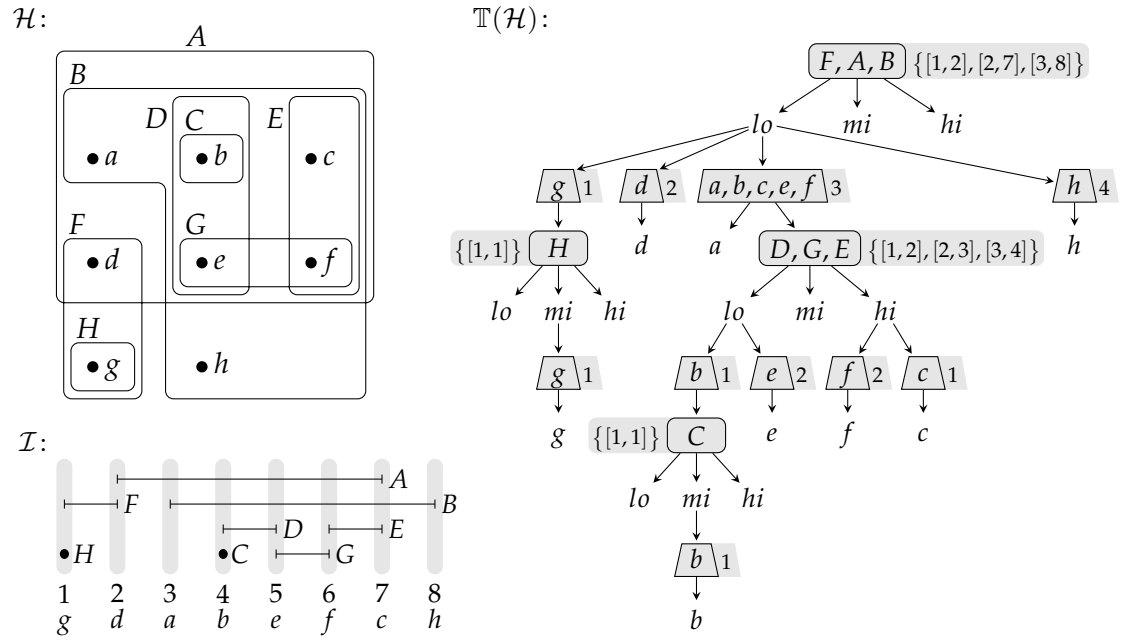

Figure 4.4: An interval hypergraph $\mathcal{H}$ and the corresponding tree representation $\mathbb{T}(\mathcal{H})$. Gray areas in $\mathbb{T}(\mathcal{H})$ indicate the color of overlap components and their slots. An overlap component $\mathcal{O}$ is represented by listing the hyperedges in $\mathcal{O}$ (sorted, for the reader's convenience, by their corresponding intervals in the canonical form of $\mathcal{O}$). The references from slot and connector nodes to their overlap components are omitted as they can be understood from the tree structure. The interval system $\mathcal{I} \cong \mathcal{H}$ can be derived from $\mathbb{T}(\mathcal{H})$ by reading the vertex nodes from left to right (see Section 4.2.3 for details).

Note that, for a mirror-symmetric overlap component $\mathcal{O}$, the orders $<_{\mathcal{O},1}$ and $<_{\mathcal{O},2}$ might be swapped, so $\mathrm{pos}_{\mathcal{O},1}$ and $\mathrm{pos}_{\mathcal{O},2}$ can exchange their values. These choices influence the construction of the tree. However, all possible $\mathbb{T}(\mathcal{H})$ are isomorphic, as the respective $\mathrm{lo}_{\mathcal{O}}$ and $\mathrm{hi}_{\mathcal{O}}$ nodes can be exchanged and the colors of the slot nodes stay the same.

The goal is to compute a canonical interval representation of $\mathcal{H}$ by applying a variant of Lindell's tree canonization algorithm [Lin92] to $\mathbb{T}(\mathcal{H})$. For this approach, it is necessary that $\mathbb{T}(\mathcal{H})$ is computable in logspace and that isomorphic interval hypergraphs have isomorphic tree representations.

Lemma 4.2.3. *For a given interval hypergraph $\mathcal{H}$, its tree representation $\mathbb{T}(\mathcal{H})$ can be computed in* FL.

Proof. Each overlap component $\mathcal{O}$ of $\mathcal{H}$ can be found in logspace by running Reingold's algorithm [Rei08] on the overlap graph $\mathbb{O}(\mathcal{H})$ and described using $O(\log n)$ bits by storing any hyperedge $B \in \mathcal{O}$. Moreover, $\mathrm{supp}(\mathcal{O})$ and the slots of $\mathcal{O}$ are easy to compute. Notice that a slot S of $\mathcal{O}$ can be identified by a tuple $\langle u, B\rangle$, where u is any vertex contained in S. Logspace computability of the relation 'located at' follows from its definition. Lemma 4.2.1 allows to compute the two slot orders $<_{\mathcal{O},1}$ and $<_{\mathcal{O},2}$ of an overlap component $\mathcal{O}$ along with the resulting interval models $\mathcal{I}_{\mathcal{O},1}$ and $\mathcal{I}_{\mathcal{O},2}$ in logspace. In particular, it can be checked whether $\mathcal{I}_{\mathcal{O},1} = \mathcal{I}_{\mathcal{O},2}$ (i.e., whether $\mathcal{O}$ is mirror-symmetric), and for each slot S of $\mathcal{O}$, the positions $\mathrm{pos}_{\mathcal{O},1}(S)$ and $\mathrm{pos}_{\mathcal{O},2}(S)$ can be computed. With this information, it is easy to compute the adjacency relation of $\mathbb{T}(\mathcal{H})$ and its vertex coloring in logspace. □

Lemma 4.2.4. *If $\mathcal{H}$ and $\mathcal{K}$ are isomorphic connected interval hypergraphs, then $\mathbb{T}(\mathcal{H}) \cong \mathbb{T}(\mathcal{K})$.*

Proof. Given an isomorphism φ from $\mathcal{H}$ to $\mathcal{K}$, it suffices to construct an isomorphism φ' from $\mathbb{T}(\mathcal{H})$ to $\mathbb{T}(\mathcal{K})$. Let $\mathcal{O} = \{B_1, \ldots, B_k\}$ be an overlap component of $\mathcal{H}$. Then $\varphi'(\mathcal{O}) = \{\varphi(B_1), \ldots, \varphi(B_k)\}$ is an overlap component of $\mathcal{K}$ isomorphic to $\mathcal{O}$; hence the nodes $\mathcal{O}$ and $\varphi'(\mathcal{O})$ get the same color in the trees $\mathbb{T}(\mathcal{H})$ and $\mathbb{T}(\mathcal{K})$. For any slot S of $\mathcal{O}$, define φ' on the slot node $S_{\mathcal{O}}$ by $\varphi'(S_{\mathcal{O}}) = \varphi(S)_{\varphi'(\mathcal{O})}$. The set $\varphi(S)$ is a slot of $\varphi'(\mathcal{O})$ because φ induces an isomorphism from $\mathcal{O}$ to $\varphi'(\mathcal{O})$. By the same reasoning, $S_{\mathcal{O}}$ and $\varphi'(S_{\mathcal{O}})$ are equally colored in $\mathbb{T}(\mathcal{H})$ and $\mathbb{T}(\mathcal{K})$. If $\mathcal{O}$ is not mirror-symmetric, let φ' take $\mathrm{lo}_{\mathcal{O}}$, $\mathrm{mi}_{\mathcal{O}}$, and $\mathrm{hi}_{\mathcal{O}}$ to $\mathrm{lo}_{\varphi'(\mathcal{O})}$, $\mathrm{mi}_{\varphi'(\mathcal{O})}$, and $\mathrm{hi}_{\varphi'(\mathcal{O})}$, respectively; otherwise φ' may swap the images of $\mathrm{lo}_{\mathcal{O}}$ and $\mathrm{hi}_{\mathcal{O}}$ in order to obey adjacency. Finally, let $\varphi'(v) = \varphi(v)$ for all $v \in \mathrm{V}(\mathcal{H})$, certainly respecting adjacency in the trees $\mathbb{T}(\mathcal{H})$ and $\mathbb{T}(\mathcal{K})$. □

4.2.3 Computing canonical interval representations

To construct a canonical interval representation for $\mathcal{H}$, the tree representation $\mathbb{T}(\mathcal{H})$ is traversed in such a way that the vertex nodes of $\mathbb{T}(\mathcal{H})$ are visited exactly in the order in which the corresponding vertices will appear in the resulting canonical interval model of $\mathcal{H}$. The tree representation (viewed as a PQ-tree) already restricts the possible traversals to those that visit the slots S of each overlap component $\mathcal{O}$ in either ascending or descending order of their position $\mathrm{pos}_{\mathcal{O},1}(S)$, where the latter is possible only if $\mathcal{O}$ is mirror-symmetric. Thus, there remains a freedom to choose one of two possible orientations for each mirror-symmetric $\mathcal{O}$. Also, for each slot S of an overlap component

of $\mathcal{H}$, the order of the vertices and overlap components located at S can be chosen freely. The generalization of Lindell's tree canonization algorithm given by Lemma 2.7.4 will be employed to come up with a canonical choice between the remaining possibilities.

Recall the logspace traversal algorithm for trees with linear order among siblings (as used in, e.g., Lindell's canonization [Lin92]). For a given node, it is possible in logspace to (1) go to its first child, (2) go to its next sibling, and (3) go to its parent. These operations allow a depth-first traversal of the tree where only the current node and whether the algorithm arrived there from its last child must be remembered. The only requirement is that the order among siblings can be evaluated in logspace. During the traversal of $\mathbb{T}(\mathcal{H})$, the following order will be used, where $\lambda_{\mathbb{T}(\mathcal{H})}$ is the canonical labeling of $\mathbb{T}(\mathcal{H})$ as obtained by Lemma 2.7.4.

- The children of an overlap component node $\mathcal{O}$ are ordered $\mathrm{lo}_{\mathcal{O}} < \mathrm{mi}_{\mathcal{O}} < \mathrm{hi}_{\mathcal{O}}$ if $\mathcal{O}$ is not mirror-symmetric or if $\lambda_{\mathbb{T}(\mathcal{H})}(\mathrm{lo}_{\mathcal{O}}) < \lambda_{\mathbb{T}(\mathcal{H})}(\mathrm{hi}_{\mathcal{O}})$; otherwise they are ordered $\mathrm{hi}_{\mathcal{O}} < \mathrm{mi}_{\mathcal{O}} < \mathrm{lo}_{\mathcal{O}}$.
- The children of the first child of an overlap component node $\mathcal{O}$ (this can be either $\mathrm{lo}_{\mathcal{O}}$ or $\mathrm{hi}_{\mathcal{O}}$) are visited in ascending order of their colors; note that these colors are all distinct.
- The children of the last child of an overlap component node $\mathcal{O}$ (this can be either $\mathrm{hi}_{\mathcal{O}}$ or $\mathrm{lo}_{\mathcal{O}}$) are visited in descending order of their colors; again, these are all distinct.
- A $\mathrm{mi}_{\mathcal{O}}$ node can have at most one child, which does not need ordering.
- The children of a slot node are ordered by the label that $\lambda_{\mathbb{T}(\mathcal{H})}$ assigns to them.

Note that ordering $\mathbb{T}(\mathcal{H})$ in this way yields a permissible realization of the PQ-tree; i.e., the slots S of each overlap component $\mathcal{O}$ are placed in ascending or descending order of their position $\mathrm{pos}_{\mathcal{O},1}(S)$. In the following, this will be exploited to compute an interval representation of the entire hypergraph $\mathcal{H}$.

Let $\rho_{\mathcal{H}} \colon \mathrm{V}(\mathcal{H}) \to \left[1, |\mathrm{V}(\mathcal{H})|\right]$ be the bijection that numbers the vertices of $\mathcal{H}$ in the same order as their nodes are visited in the described traversal of $\mathbb{T}(\mathcal{H})$; it clearly can be computed in logspace.

Lemma 4.2.5. *The function $\rho_{\mathcal{H}}$ is an interval representation of $\mathcal{H}$.*

Proof. Let $B \in \mathcal{H}$ be any hyperedge, it must be shown that its image $\rho_{\mathcal{H}}(B)$ is an interval. Let $\mathcal{O}$ be the overlap component that contains B. As the slot nodes of $\mathcal{O}$ are visited in either ascending or descending order of their appearance in the interval model $\mathcal{I}_{\mathcal{O},1}$ corresponding to $<_{\mathcal{O},1}$, the slots of $\mathcal{O}$ that are contained in B are visited consecutively, and with them also the vertices contained in them, as these are exactly the vertices contained in the subtrees rooted at these slot nodes. □

Now we are ready to prove the main result on interval hypergraphs.

Theorem 4.2.6. *Canonical interval representations of interval hypergraphs can be computed in* FL.

Proof. Given an interval hypergraph $\mathcal{H}$, the algorithm computes the interval representation $\rho_{\mathcal{H}}$ of $\mathcal{H}$ defined above. By Lemma 2.7.4, the canonical labeling $\lambda_{\mathbb{T}(\mathcal{H})}$ of $\mathbb{T}(\mathcal{H})$ can be computed in logspace; the rest of the computation of $\rho_{\mathcal{H}}$ is possible in logspace as well.

It remains to show that the representations $\rho_{\mathcal{H}}$ and $\rho_{\mathcal{K}}$ of isomorphic interval hypergraphs $\mathcal{H} \cong \mathcal{K}$ map these graphs to the same interval model $\rho_{\mathcal{H}}(\mathcal{H}) = \rho_{\mathcal{K}}(\mathcal{K})$. By Lemma 4.2.4, the colored trees $\mathbb{T}(\mathcal{H})$ and $\mathbb{T}(\mathcal{K})$ are isomorphic. Hence, the canonical labelings $\lambda_{\mathbb{T}(\mathcal{H})}$ and $\lambda_{\mathbb{T}(\mathcal{K})}$ map these trees to the same colored tree $\lambda_{\mathbb{T}(\mathcal{H})}\big(\mathbb{T}(\mathcal{H})\big) = \lambda_{\mathbb{T}(\mathcal{K})}\big(\mathbb{T}(\mathcal{K})\big)$. As the interval model $\rho_{\mathcal{H}}(\mathcal{H})$ depends only on the tree $\lambda_{\mathbb{T}(\mathcal{H})}\big(\mathbb{T}(\mathcal{H})\big)$, this implies $\rho_{\mathcal{H}}(\mathcal{H}) = \rho_{\mathcal{K}}(\mathcal{K})$. □

Translated into the language of matrices, Theorem 4.2.6 has algorithmic consequences for testing the consecutive-ones property that was defined in Section 1.2.

Corollary 4.2.7. *There is a logspace algorithm that decides whether a given boolean matrix has the consecutive-ones property and computes an appropriate permutation of the columns if this is the case.*

4.3 Canonizing interval graphs and convex graphs

The reduction of Lemma 4.1.4 transforms Theorem 4.2.6 into a result for interval graphs.

Corollary 4.3.1. *Canonical interval representations of interval graphs can be computed in* FL.

A bipartite graph G is called *convex* if one of its vertex classes can be linearly ordered so that the neighborhoods of the vertices in the other class are intervals with respect to this order. If both vertex classes have such orderings, G is called *biconvex*.

The *incidence graph* of a hypergraph $\mathcal{H}$ is the bipartite graph with vertex classes $\mathrm{V}(\mathcal{H})$ and $\mathcal{H}$, where two vertices $v \in \mathrm{V}(\mathcal{H})$ and $A \in \mathcal{H}$ are adjacent if and only if $v \in \mathcal{H}$. Note that this transformation yields exactly the convex graphs when applied to interval hypergraphs.

Corollary 4.3.2. *Canonical labelings of convex graphs can be computed in* FL.

Proof. Let G be a connected convex graph with vertex classes U and V. Reversing the aforementioned transformation of hypergraphs to their incidence graphs results in two hypergraphs $\mathcal{H}_U = \big(U, \{\!\{\mathrm{N}(v) \mid v \in V\}\!\}\big)$ and $\mathcal{H}_V = \big(V, \{\!\{\mathrm{N}(u) \mid u \in U\}\!\}\big)$. Since G is convex, at least one of them is an interval hypergraph. Suppose that this is true for $\mathcal{H}_U$ and let $\rho_{\mathcal{H}_U}$ be its canonical interval representation. The representation $\rho_{\mathcal{H}_U}$ can also be viewed as a labeling of U, and sorting the labels $\rho_{\mathcal{H}_U}\big(\mathrm{N}(v)\big)$, $v \in V$, results in a labeling of V. Define λ_U as the combination of these labelings:

$$\lambda_U(v) = \begin{cases} \rho_{\mathcal{H}_U}(v) & \text{if } v \in U \\ \big|U\big| + \big|\big\{u \in V \,\big|\, \rho_{\mathcal{H}_U}\big(\mathrm{N}(u)\big) \leq \rho_{\mathcal{H}_U}\big(\mathrm{N}(v)\big)\big\}\big| & \text{if } v \in V \end{cases}$$

If $\mathcal{H}_V$ is also an interval hypergraph (i.e., if G is biconvex), define λ_V analogously, compare the resulting graphs $\lambda_U(G)$ and $\lambda_V(G)$ lexicographically, and choose the labeling with the smaller image as the canonical labeling λ_G. Otherwise, let $\lambda_G = \lambda_U$.

If the graph G is disconnected, split it into connected components using Reingold's algorithm [Rei08], canonize each of the components, and compose the total labeling of G componentwise in lexicographic order. □

Another generalization of interval graphs is the class of *rooted directed path graphs*, i.e., intersection graphs of paths in a rooted directed tree. For this class, only polynomial-time isomorphism algorithms are known [BPT96; EPT00]. While maxcliques in a rooted directed path graph can still be recognized in a way similar to that of Lemma 4.1.2, the procedure for linearly ordering maxcliques as given in Lemma 4.2.1 cannot be employed in the presence of tree nodes of degree more than 2.

In the above paragraph, it is important that trees are rooted and directed accordingly, as intersection graphs of paths in unrooted directed trees are isomorphism-complete [BPT96].

4.4 Computing proper and unit interval representations

Let $\mathcal{I} = \{[a_i, b_i] \mid 1 \leq i \leq n\}$ be a proper interval system. As there are no inclusions, the start points a_i are pairwise distinct, and the same is true about the end points b_i. Suppose that $a_i < a_{i+1}$ for all $i < n$. As $\mathcal{I}$ is proper, it follows that also $b_i < b_{i+1}$. This yields a natural geometric order on $\mathcal{I}$.

Now let G be a proper interval graph. Any proper interval representation $\alpha\colon \mathrm{V}(G) \to \mathcal{I}$ of G induces the *geometric order* $<_\alpha$ on $\mathrm{V}(G)$ defined by

$$u <_\alpha v \Leftrightarrow \alpha(u) < \alpha(v).$$

Observe that, if u is adjacent to v with $u <_\alpha v$, then u is adjacent to all w with $u <_\alpha w \leq_\alpha v$. This implies that each $\mathrm{N}[u]$ is an interval with respect to $<_\alpha$; therefore, $\mathcal{N}[G]$ is an interval hypergraph. As mentioned in the introduction, the converse is also true: If $\mathcal{N}[G]$ is an interval hypergraph, then G is a proper interval graph [Rob69].

It is clear that $\mathcal{N}[G] \cong \mathcal{N}[H]$ whenever $G \cong H$. Harary and McKee [HM96] show that the converse is also true if G is chordal. Combining this with Theorem 4.2.6 immediately gives a logspace computable complete invariant for proper interval graphs: G is isomorphic to another graph G' exactly when the canonical interval models of $\mathcal{N}[G]$ and $\mathcal{N}[G']$ are equal.

Note that an interval model of the hypergraph $\mathcal{N}[G]$ is generally not an interval model of the graph G. Note also that the minimal interval model constructed from the maxclique bundle hypergraph $\mathcal{B}(G)$ (as in Corollary 4.3.1) is not proper if the graph contains at least one edge; see Figure 4.5 for an example.

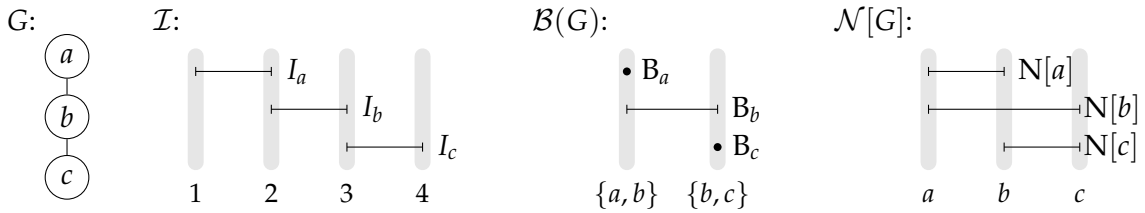

Figure 4.5: A proper interval graph G and a proper interval model $\mathcal{I}$ of G. However, the bundle hypergraph $\mathcal{B}(G)$ is not proper. The neighborhood hypergraph $\mathcal{N}[G]$ is neither proper nor an intersection model of G.

4.4.1 Computing proper interval representations

In this section, it is shown that canonical proper interval representations of proper interval graphs can be computed in logspace. While this also follows from the results in Section 4.6, looking at this special case separately is instructive, especially when comparing the techniques used here with those used for proper circular-arc graphs in Section 5.3.

The following fact on overlap-connected hypergraphs will be useful.

Lemma 4.4.1. *Let $\mathcal{F}$ and $\mathcal{E}$ be overlap-connected hypergraphs with* $\operatorname{supp}(\mathcal{F}) = \operatorname{supp}(\mathcal{E})$*, each containing at least two different hyperedges. Then their union $\mathcal{F} \cup \mathcal{E}$ is overlap-connected, too.*

Proof. Choose $F \in \mathcal{F}$ and $E \in \mathcal{E}$ with nonempty intersection. If $F \between E$ or $F = E$, the claim is obviously true. Otherwise, suppose that $E \subsetneq F$. Note that $F \neq \operatorname{supp}(\mathcal{F})$ because F is not the only hyperedge in $\mathcal{F}$ and $\mathbb{O}(\mathcal{F})$ is connected. Let $x \in \operatorname{supp}(\mathcal{F}) \setminus F$. Since $\operatorname{supp}(\mathcal{F}) = \operatorname{supp}(\mathcal{E})$, there is a hyperedge $E' \in \mathcal{E}$ containing x. By the connectedness of $\mathbb{O}(\mathcal{E})$, there is an overlap-path $E_1 \between E_2 \between \dots \between E_l$ connecting $E = E_1$ and $E' = E_l$. Let $m < l$ be the largest index such that $E_m \subseteq F$. Then $E_{m+1} \between F$. □

It is easy to see that a proper interval graph with n vertices always has a *sharp* proper interval model $\mathcal{I} = \big\{[a_i, b_i] \bigm| 1 \leq i \leq n\big\}$, i.e., one where $\{a_i, b_i \mid 1 \leq i \leq n\} = \{1, 2, \dots, 2n\}$. In this section, only such interval models will be considered. Together with a proper interval representation $\alpha\colon \mathrm{V}(G) \to \mathcal{I}$, the graph G has also the reflected proper interval representation $\alpha^*\colon \mathrm{V}(G) \to \mathcal{I}^*$ defined by $\alpha^*(v) = r\big(\alpha(v)\big)$, where r is the mirror reflection $r(x) = 2n + 1 - x$. The geometric orders induced by α and α^* are the reversal of each other, i.e., $<_{\alpha^*} = (<_\alpha)^{-1}$. The first, graph-theoretic part of the following lemma restates a result by Deng, Hell, and Huang [DHH96, Corollary 2.5] in a different form and is proved in a different way, which allows to obtain also a logspace computability result.

Lemma 4.4.2. *Let G be a connected proper interval graph without twins. Then, up to reflection, G has a unique proper interval representation. The latter is computable in logspace.*

Proof. Let $\alpha\colon \mathrm{V}(G) \to \mathcal{I}$ be a proper interval representation of G, and let $<_\alpha$ be the associated geometric order on $\mathrm{V}(G) = \{v_1, \dots, v_n\}$. For the purposes of this proof, suppose that $v_1 <_\alpha \cdots <_\alpha v_n$. As mentioned before, $<_\alpha$ defines an interval representation of the neighborhood hypergraph $\mathcal{N}[G]$.

Given a nonuniversal vertex v_i, let $s = s(i)$ be the largest index such that $s < i$ and $v_s \notin \mathrm{N}[v_i]$, and, similarly, let $t = t(i)$ be the smallest index such that $t > i$ and $v_t \notin \mathrm{N}[v_i]$. At least one of these indices is well defined. Note that $\mathrm{N}[v_i] \between \mathrm{N}[v_s]$. Indeed, $v_s \notin \mathrm{N}[v_i]$, $v_i \notin \mathrm{N}[v_s]$, and v_{s+1} belongs to both sets (v_s and v_{s+1} are adjacent because G is connected). A symmetric argument shows $\mathrm{N}[v_i] \between \mathrm{N}[v_t]$. Note that neither v_s nor v_t is universal. It follows that, for any nonuniversal v_i, there is a subsequence of indices $i_1, \dots, i_k$ containing i such that

$$\mathrm{N}[v_{i_1}] \between \mathrm{N}[v_{i_2}] \between \cdots \between \mathrm{N}[v_{i_k}] \quad \text{and} \quad \mathrm{N}[v_{i_1}] \cup \mathrm{N}[v_{i_2}] \cup \cdots \cup \mathrm{N}[v_{i_k}] = \mathrm{V}(G).$$

If there is a universal vertex u (there can be at most one as there are no twins), remove the hyperedge $\mathrm{N}[u]$ from $\mathcal{N}[G]$ and denote the modified hypergraph by $\mathcal{N}'[G]$. It follows

with Lemma 4.4.1 that $\mathcal{N}'[G]$ is overlap-connected. By Lemma 4.2.1 there is a unique pair of mutually reversed strict orders $<$ and $<^*$ on the slots of $\mathcal{N}'[G]$ such that the slots appear according to one of these orders in any interval representation of the hypergraph.

Since G has no twins, the slots of $\mathcal{N}[G]$ are singletons $\{v_1\}, \ldots, \{v_n\}$. Note that $\mathcal{N}'[G]$ has the same slots as $\mathcal{N}[G]$. Thus, $<$ and $<^*$ can be viewed as orders on $V(G)$, and one of them must coincide with $<_\alpha$. To prove the uniqueness result, it now suffices to notice that $<_\alpha$, i.e., the sequence $v_1, \ldots, v_n$, uniquely determines α. Indeed, it follows that $\alpha(v_i) = [a_i, b_i]$, where

$$\begin{aligned} a_i &= i + \left|\{j < i \mid v_j \text{ is nonadjacent to } v_i\}\right| \\ \text{and } \; b_i &= a_i + 1 + \deg(v_i). \end{aligned} \tag{4.2}$$

Finally, note that equation (4.2) allows to compute α in logspace once a permissible slot order of $\mathcal{N}'[G]$ is known. The latter can be computed in logspace by Lemma 4.2.1. □

Theorem 4.4.3. *Canonical proper interval representations for proper interval graph can be computed in* FL.

Proof. Let G be a proper interval graph. First assume that G is connected. If G has no twins, Lemma 4.4.2 allows to compute two mutually reflected proper interval representations $\alpha\colon V(G) \to \mathcal{I}$ and $\alpha^*\colon V(G) \to \mathcal{I}^*$. Choose α as canonical if $\mathcal{I} < \mathcal{I}^*$ (w.r.t. the lexicographic order as defined at the beginning of Chapter 2); otherwise α^* is chosen (if $\mathcal{I} = \mathcal{I}^*$, either choice is good). If G has twins, there still is a canonical pair $\{\alpha, \alpha^*\}$ which is unique up to interchanging labels within a twin class. In order to compute this pair, replace each twin class by a single representative, obtaining a twin-free quotient graph G'. As in the proof of Lemma 4.4.2, compute the proper ordering $v'_1, \ldots, v'_{n'}$ on $V(G')$ (unique up to reflection). Further, expand this sequence by substituting each v'_i with all the twins it represents, obtaining an ordering $v_1, \ldots, v_n$ of $V(G)$. Finally, the intervals $\alpha(v_i) = [a_i, b_i]$ are computed according to equation (4.2). Another candidate is $\alpha^* = r \circ \alpha$; choose one of the two which gives a $<$-least interval model.

If G is disconnected, split it into connected components $G_1, \ldots, G_k$ using Reingold's reachability algorithm [Rei08]. For each of them, compute the canonical representation $\alpha_{G_j}\colon V(G_j) \to \mathcal{I}_j$ and sort the interval models $\mathcal{I}_1, \ldots, \mathcal{I}_k$. Then merge the representations α_{G_j} into an integrated representation $\alpha_G\colon V(G) \to \mathcal{I}$ so that the supports of the interval models $\mathcal{I}_j$ appear in $\operatorname{supp}(\mathcal{I})$ according to the order established by sorting. □

Both the linear-time [DHH96; HSS01] and the AC^2 [BHI07] representation algorithms for proper interval graphs are based on computing the canonical order of vertices of the input graph as in the proof of Lemma 4.4.2; as already mentioned, the combinatorial part of this lemma is proved by Deng, Hell, and Huang [DHH96] in a different language and by a different argument.

4.4.2 Computing unit interval representations

Concluding the section on proper interval graphs, let us turn to the task of finding a canonical unit interval representation. A graph is a unit interval graph if it has an interval model over rationals in which every interval has unit length. It is well known that the class of proper interval graphs is equal to the class of unit interval graphs [Rob69].

Corollary 4.4.4. *Canonical unit interval representations of proper interval graphs can be computed in* FL.

Proof. Given a unit interval graph G, Theorem 4.4.3 allows to compute canonical proper interval representation α_G for G in logspace. We may assume that G is connected; if it is not, we can deal with the connected components individually and piece them back together in the end. Let $v_1, \ldots, v_n$ be the vertices of G in the geometric ordering induced by α_G. For every vertex $v \neq v_1$, let l_v be the least neighbor of v in this ordering. The edge (l_v, v) is called *principal edge at* v. It is easy to see that the set of principal edges forms a directed tree T on $\mathrm{V}(G)$ that is rooted at v_1. Order the children of each vertex in T according to the above ordering.

For a vertex v, let $k(v)$ be the level at which v is located in T, and let $p(v)$ be the number assigned to v in the postorder traversal of T. Since T is constructible in logspace, both values can be computed in FL for any $v \in \mathrm{V}(G)$. Assign to each vertex v the value $v^L = k(v) + p(v)/n$. Corneil et al. show that assigning $\left[v_i^L, v_i^L + 1\right]$ to v_i for every $i \in [1, n]$ yields a unit interval representation for G [CKN+95, Theorem 3.2].

Since α_G is a canonical proper interval representation of G and the procedure does not involve any arbitrary choices, it computes a canonical unit interval representation for G in FL. □

4.5 Constrained interval and intersection lengths

Let G be an interval graph and let $\ell\colon \mathrm{V}(G) \to \mathbb{N}^+$ and $s\colon \mathrm{E}(G) \to \mathbb{N}^+$ be functions that prescribe the lengths of intervals and pairwise intersections, respectively. In this section, it is shown how to construct (ℓ, s)-respecting interval representations in linear time or alternatively in logspace, and s-respecting interval representations in $O(nm)$ time or alternatively in logspace.

The first step towards these algorithms is to show that all interval representation of the appropriate type have the same inclusion and overlap relationships, and that these relations can be computed efficiently when G, ℓ (and s) are given as input. This is described in Section 4.5.1.

The results on (ℓ, s)-respecting interval representations (which are in Section 4.5.2) are obtained in a similar way as those for interval hypergraphs in Section 4.2: First, graphs with overlap-connected representations are considered. Their representations are unique up to reflection and can be computed efficiently (if they exist). For graphs with several overlap components, these components can be arranged into a tree, and their (ℓ, s)-respecting interval representations can be combined into one for the whole graph. It is also shown that all (ℓ, s)-respecting interval representations of G are isomorphic, giving an alternative proof for a result of Fulkerson and Gross [FG65, Theorem 2.1].

In Section 4.5.3, it is shown how s-respecting interval representations can be computed efficiently. To obtain this result, the algorithm for computing an (ℓ, s)-respecting interval representation is repeatedly used as a subroutine. It is shown that the lengths of the pairwise intersections already determine the interval lengths if the resulting s-respecting interval representation is required to be minimal, i.e., if it contains a minimum number of points. It is also shown that all minimal s-respecting interval representations are isomorphic.

Section 4.5.4 is concerned with the variant of the interval representation problem for which Pe'er and Shamir gave a polynomial-time algorithm [PS97]: On the one hand, the bundle hypergraph of the input graph is required to have a unique interval representation (up to reflection), and on the other hand, general lower and upper bounds on distances between extreme points of intervals are allowed. In Section 4.5.4, it is shown that this variant is in fact NL-complete. Thus it is unlikely that this generalization of the ℓ-respecting interval representation problem is solvable in deterministic logspace even for this restricted class of input graphs.

4.5.1 Deriving structural information

Let G be a graph, define $n = |V(G)|$ and $m = |E(G)|$, and let $\ell\colon V(G) \to \mathbb{N}^+$ and $s\colon E(G) \to \mathbb{N}^+$ specify the desired interval and intersection lengths. For convenience, define $s(u,v) = s\big(\{u,v\}\big)$ for $\{u,v\} \in E(G)$, and define $s(u,v) = 0$ for $\{u,v\} \notin E(G)$. Using this convention, define two relations $R_{\ell,s}, R_s \subseteq V(G) \times V(G)$:

$$(u,v) \in R_{\ell,s} \Leftrightarrow \{u,v\} \in E(G) \wedge \ell(u) > s(u,v)$$
$$(u,v) \in R_s \;\Leftrightarrow \{u,v\} \in E(G) \wedge \exists w \in V(G) \setminus \{u,v\} : s(w,u) > \min\big\{s(w,v), s(u,v)\big\}$$

By the following lemma, these relations characterize a structural property that *all* (ℓ,s)-respecting (resp., minimal s-respecting) interval representations of G have in common.

Lemma 4.5.1. *Let G be a graph, let $\{u,v\} \in E(G)$ be an edge, and let $\alpha\colon V(G) \to \mathcal{I}$ be an interval representation of G.*

(a) *If α is (ℓ,s)-respecting, then $\alpha(u) \setminus \alpha(v) \neq \emptyset$ if and only if $(u,v) \in R_{\ell,s}$.*

(b) *If α is minimally s-respecting, then $\alpha(u) \setminus \alpha(v) \neq \emptyset$ if and only if $(u,v) \in R_s$.*

Proof. Part (a) follows directly from the definitions (for the first equivalence, $\{u,v\} \in E(G)$ is required):

$$(u,v) \in R_{\ell,s} \Leftrightarrow \ell(u) > s(u,v) \Leftrightarrow |\alpha(u)| > |\alpha(u) \cap \alpha(v)| \Leftrightarrow \alpha(u) \setminus \alpha(v) \neq \emptyset$$

To show part (b), note that $(u,v) \in R_s$ means by definition that there is a $w \in V(G)$ such that $s(w,u) > s(w,v)$ or $s(w,u) > s(u,v)$. Either way, there must be a point $p \in \alpha(w) \cap \big(\alpha(u) \setminus \alpha(v)\big)$, implying $\alpha(u) \setminus \alpha(v) \neq \emptyset$.

For the backward direction, consider a point $p \in \alpha(u) \setminus \alpha(v)$. By minimality of α, there is a vertex $w \in V(G) \setminus \{u\}$ with $p \in \alpha(w)$. Note that $w \neq v$ by the choice of p. If $\alpha(w) \supset \alpha(u) \cap \alpha(v)$ (see Figure 4.6(a)), it follows that $s(w,u) > s(u,v)$. Otherwise (see Figure 4.6(b)), $\alpha(w) \cap \alpha(u) \supsetneq \alpha(w) \cap \alpha(v)$, and thus $s(w,u) > s(w,v)$. □

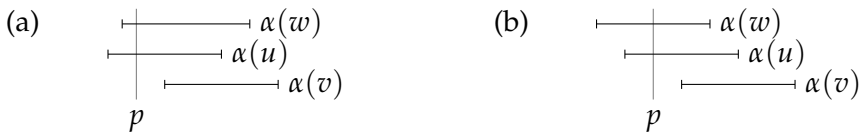

Figure 4.6: The two cases in the proof of Lemma 4.5.1(b)

Lemma 4.5.2. *$R_{\ell,s}$ and R_s can be enumerated in time $O(m)$ and $O(nm)$, respectively, and both can be enumerated in logspace.*

Proof. Both $R_{\ell,s}$ and R_s are defined by first order formulas; these can easily be evaluated in logspace. Iterating over all pairs $(u,v) \in \mathrm{V}(G) \times \mathrm{V}(G)$ gives logspace enumeration.

To enumerate $R_{\ell,s}$ in $O(m)$ time, loop over all edges $\{u,v\} \in \mathrm{E}(G)$ (considering both orientations) and output (u,v) if $\ell(u) > s(u,v)$.

To enumerate R_s in $O(nm)$ time, loop over all edges $\{w,u\} \in \mathrm{E}(G)$ (again, considering both orientations) and all nodes $v \in \mathrm{V}(G) \setminus \{w,u\}$, and output (u,v) if $s(w,u) > \min\{s(w,v), s(u,v)\}$. □

Next, define the relations $\between_{\ell,s}$, $\subseteq_{\ell,s}$, $\between_s$, and $\subseteq_s$ on $\mathrm{V}(G) \times \mathrm{V}(G)$ by

$$u \between_{\ell,s} v \Leftrightarrow (u,v) \in R_{\ell,s} \wedge (v,u) \in R_{\ell,s} \qquad u \subseteq_{\ell,s} v \Leftrightarrow \{u,v\} \in \mathrm{E}(G) \wedge (u,v) \notin R_{\ell,s}$$
$$u \between_s \; v \Leftrightarrow (u,v) \in R_s \; \wedge (v,u) \in R_s \qquad u \subseteq_s \; v \Leftrightarrow \{u,v\} \in \mathrm{E}(G) \wedge (u,v) \notin R_s$$

By Lemma 4.5.1, these relations describe the situation in any appropriate representation of G, e.g. $u \between_{\ell,s} v \Leftrightarrow \alpha(u) \between \alpha(v)$ holds in any (ℓ,s)-respecting interval representation α of G, and $u \between_s v \Leftrightarrow \alpha'(u) \between \alpha'(v)$ holds in any minimal s-respecting interval representation α' of G.

Lemma 4.5.3. *Let $\alpha\colon \mathrm{V}(G) \to \mathcal{I}$ be any s-respecting interval representation of G. For any three vertices $v, w_1, w_2 \in \mathrm{V}(G)$ such that $\alpha(w_1) \between \alpha(v) \between \alpha(w_2)$, the intervals $\alpha(w_1)$ and $\alpha(w_2)$ overlap $\alpha(v)$ from the same side if and only if $s(w_1,w_2) > \min\{s(w_1,v), s(w_2,v)\}$.*

Note that this condition can be decided both in constant time and in logspace.

Proof. See Figure 4.7 for an illustration. If $\alpha(w_1)$ and $\alpha(w_2)$ overlap $\alpha(v)$ from the same side, then $\alpha(w_1)$ and $\alpha(w_2)$ contain at least one common point outside $\alpha(v)$, making their intersection larger than the minimum of $|\alpha(w_1) \cap \alpha(v)|$ and $|\alpha(w_2) \cap \alpha(v)|$.

Now suppose to the contrary that $\alpha(w_1)$ and $\alpha(w_2)$ overlap $\alpha(v)$ from different sides. In this case $\big(\alpha(w_1) \cap \alpha(v)\big) \setminus \alpha(w_2)$ and $\big(\alpha(w_2) \cap \alpha(v)\big) \setminus \alpha(w_1)$ are both nonempty, implying that $|\alpha(w_1) \cap \alpha(w_2)|$ is smaller than both $|\alpha(w_1) \cap \alpha(v)|$ and $|\alpha(w_2) \cap \alpha(v)|$. □

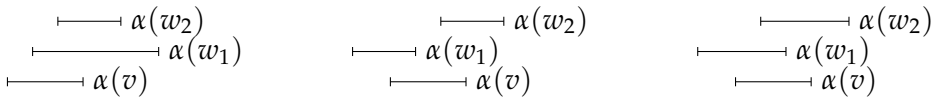

Figure 4.7: Proof of Lemma 4.5.3: Different ways how $\alpha(w_1)$ and $\alpha(w_2)$ can overlap $\alpha(v)$.

4.5.2 Given interval and intersection lengths

Let G be a graph, and let the functions $\ell\colon \mathrm{V}(G) \to \mathbb{N}^+$ and $s\colon \mathrm{E}(G) \to \mathbb{N}^+$ specify the desired interval and intersection lengths, respectively. In this section, a linear-time and a logspace algorithm are presented that both construct an (ℓ,s)-respecting interval representation of G, or detect that such a representation does not exist.

Define the graph $G_{\ell,s}$ on the same vertex set as G by $\mathrm{E}(G_{\ell,s}) = \big\{\{u,v\} \in \mathrm{E}(G) \,\big|\, u \between_{\ell,s} v\big\}$, and call the connected components of $G_{\ell,s}$ the *overlap components* of G.

The next lemma shows that (ℓ, s)-respecting interval representations of $\between_{\ell,s}$-connected graphs are unique up to reflection, just as Lemma 4.2.1 shows that the interval models of $\between$-connected hypergraphs are unique up to reflection. Indeed, the proofs use the same ideas.

Lemma 4.5.4. *Given a graph G and two functions $\ell\colon \mathrm{V}(G) \to \mathbb{N}^+$ and $s\colon \mathrm{E}(G) \to \mathbb{N}^+$ such that $G_{\ell,s}$ is connected, it is possible in linear time (resp., in logspace) to compute an (ℓ, s)-respecting interval representation $\alpha\colon \mathrm{V}(G) \to \mathcal{I}$ of G, or to detect that none exists. Moreover, $\mathcal{I}$ is unique up to reflection.*

Proof. Let $v_1, v_2, \ldots, v_N$ be a walk in $G_{\ell,s}$ that visits every vertex at least once; such a walk can be constructed in linear time using depth first search or in logspace using Reingold's universal exploration sequences [Rei08]. The following algorithm computes an interval representation $\alpha\colon \mathrm{V}(G) \to \mathcal{I}$ of G by moving along this walk. At each step of the walk, compute an interval $I_k = [p_k, p_k + \ell(v_k) - 1]$ for v_k. After that, let $\alpha(v_k) = I_k$, if there is no $j < k$ with $v_j = v_k$. Define $p_1 = 1$ and $p_2 = \ell(v_1) - s(v_1, v_2) + 1$. Note that after I_1 has been placed, there are only two possibilities for I_2 that respect (ℓ, s); see Figure 4.8 for an illustration. After that, all further intervals are completely determined because Lemma 4.5.3 allows to detect whether the next interval I_k has to overlap I_{k-1} from the same side as I_{k-2}. This allows to define

$$p_k = \begin{cases} p_{k-1} - \ell(v_k) + s(v_{k-1}, v_k) & \text{if } I_k \text{ overlaps } I_{k-1} \text{ from the left, and} \\ p_{k-1} + \ell(v_{k-1}) - s(v_{k-1}, v_k) & \text{otherwise.} \end{cases}$$

Note that I_k can be computed from the walk and the functions ℓ and s, remembering only the two previous intervals; thus α can be obtained in logspace.

In a post-processing step, check that α is indeed an (ℓ, s)-respecting interval representation of G. Additionally, shift the resulting intervals such that 1 becomes the smallest point.

The uniqueness up to reflection follows from the fact that the only arbitrary decision (except shifting) was to place $\alpha(v_2)$ right of $\alpha(v_1)$. □

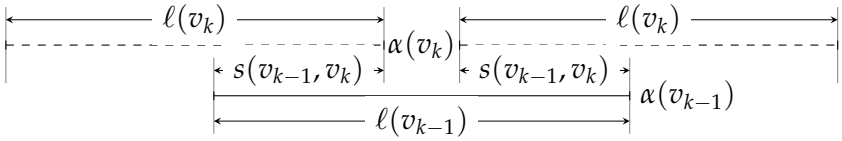

Figure 4.8: Proof of Lemma 4.5.4: If $v_k \between_{\ell,s} v_{k-1}$, and if $\alpha(v_{k-1})$ is already determined, there remain only the two dashed possibilities for $\alpha(v_k)$.

The next step is to generalize Lemma 4.5.4 to the case that $G_{\ell,s}$ is not connected. We can assume that there are no vertices v and v' such that both $v \subseteq_{\ell,s} v'$ and $v' \subseteq_{\ell,s} v$ hold; otherwise compute an (ℓ, s)-respecting interval representation α for $G \setminus \{v'\}$ and extend it by $v' \mapsto \alpha(v)$ afterwards. Let $\mathcal{C} = \{G_1, \ldots, G_k\}$ be the connected components of $G_{\ell,s}$. Define the binary relation $\leq_{\ell,s}$ on $\mathcal{C}$ that has $G_i \leq_{\ell,s} G_j$ if $i = j$ or if there are vertices $u \in \mathrm{V}(G_i)$ and $v \in \mathrm{V}(G_j)$ such that $u \subseteq_{\ell,s} v$. The latter implies that, for any (ℓ, s)-respecting interval representation α of G, the interval $\bigcup_{u \in G_i} \alpha(u)$ is contained in

some slot $S \subseteq \alpha(v)$ of $\alpha(G_j)$, because otherwise there would be an overlap-path from $\alpha(G_i)$ to $\alpha(G_j)$. Thus $\leq_{\ell,s}$ is a partial order on the overlap components of G, and each overlap component has a unique smallest successor. If G is connected, the graph $(\mathcal{C}, \leq_{\ell,s})$ is also connected; removing reflexive and transitive edges results in a rooted tree $T_{\ell,s}$, which will be called the *overlap component tree* of G.

Theorem 4.5.5. *Given a graph G and two functions $\ell\colon V(G) \to \mathbb{N}^+$ and $s\colon E(G) \to \mathbb{N}^+$, it is possible in linear time (resp., logspace) to compute an (ℓ, s)-respecting interval representation $\alpha\colon V(G) \to \mathcal{I}$ of G, or to detect that none exists. Moreover, for any (ℓ, s)-respecting interval representation $\alpha'\colon V(G) \to \mathcal{I}'$ of G, there is a hypergraph isomorphism φ from $\mathcal{I}$ to $\mathcal{I}'$ with $\varphi(\alpha(v)) = \alpha'(v)$ for all $v \in V(G)$.*

Proof. We may assume that G is connected, otherwise consider its connected components separately and concatenate their representations afterwards.

The algorithm works as follows: As pre-processing steps, it computes the connected components $\mathcal{C} = \{G_1, \ldots, G_k\}$ of $G_{\ell,s}$, an (ℓ, s)-respecting interval representation α_i for each $G_i \in \mathcal{C}$, and the overlap component tree $T_{\ell,s}$. The main part of the algorithm constructs an (ℓ, s)-respecting interval representation of G by combining appropriately shifted copies of the representations of the overlap components. This is done in a depth-first traversal of the overlap component tree. Each overlap component G_i is assigned an offset o_i by which its representation is shifted. The representation of the root component G_r is not shifted, i.e., $o_r = 0$. For each other overlap component G_i, compute the offset o_i as the sum $o_i = o_j + n_i + s_i$, where

- o_j is the offset of the parent G_j of G_i,
- n_i is the number of points in the interval model $\alpha_j(G_j)$ that are to the left of the slot S that should contain the image of G_i according to $\subseteq_{\ell,s}$, and
- s_i is the sum of the model sizes of all previously visited siblings of G_i, for which $\subseteq_{\ell,s}$ indicates that their images should be contained in the same slot S of $\alpha(G_j)$ as the image of G_i.

The model sizes n_i and s_i are available from the interval representations of the overlap components. Figure 4.9 shows an example. For a vertex $v \in V(G)$, let G_{i_v} be the overlap component that contains v. Then the algorithm checks whether $\alpha(v) = \alpha_{i_v}(v) + o_{i_v}$ is an (ℓ, s)-respecting interval representation of G and outputs α if this is the case.

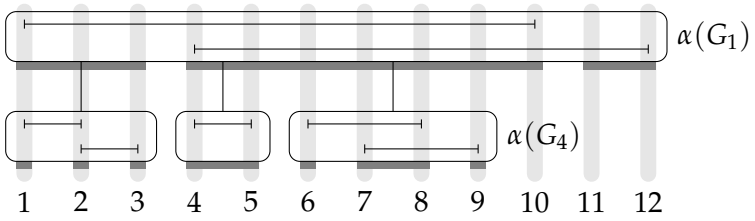

Figure 4.9: Proof of Theorem 4.5.5: The offset o_4 for the overlap component G_4 is the sum of the offset $o_1 = 0$ of its parent G_1, the number $n_4 = \left|\{[1,3]\}\right| = 3$ of points in $\alpha_1(G_1)$ left of the slot $S = [4, 10]$ that should contain the image of G_4, and the sum $s_4 = 2$ of model sizes of previously handled siblings of G_4 that are contained in the same slot, resulting in $o_4 = 5$.

If G admits an (ℓ, s)-respecting interval representation, then this algorithm will find one: Each component has a unique representation up to reflection by Lemma 4.5.4, implying that they all have the same length; and in every (ℓ, s)-respecting interval representation of G, each overlap component must be placed in the appropriate slot of its parent overlap component.

In the construction of the representation α, the only arbitrary choices are the precise placement of overlap components within the slot of their parent component that contains them, the order of the connected components of G, and whether the representations of the individual overlap components are reflected. All these choices can be transformed into one another by isomorphisms of the resulting interval system $\mathcal{I}$, which is thus unique up to isomorphism. Moreover, for any (ℓ, s)-respecting interval representation $\alpha' \colon \mathrm{V}(G) \to \mathcal{I}'$ of G, modifying the choices that result in α to match the choices that result in α' specifies a hypergraph isomorphism φ from $\mathcal{I}$ to $\mathcal{I}'$ with $\varphi\big(\alpha(v)\big) = \alpha'(v)$ for all $v \in \mathrm{V}(G)$.

It remains to show that the algorithm can be implemented in linear time or logspace. Connected components can be found in linear time using depth first search, and in logspace using Reingold's connectivity algorithm [Rei08]. The (ℓ, s)-respecting interval representations of the overlap components can be computed using Lemma 4.5.4. The construction of the overlap component tree $T_{\ell,s}$ can easily be implemented in logspace. To obtain this tree in linear time, compute $\leq_{\ell,s}$ by iterating over the edges of G, and remove reflexive and transitive arcs; see [HMR93, Proposition 3.6] for how the latter is possible in linear time. Computing the offsets for shifting is clearly possible in linear time. To compute them in logspace, traverse the overlap component tree (see e.g. [Lin92] for how to do this in logspace). During this traversal, store only the shift-offset o_i for the current overlap component G_i; this is sufficient because the offset $o_j = o_i - n_i - s_i$ of the parent component G_j of G_i can be recovered from o_i by recomputing the numbers n_i and s_i. □

4.5.3 Given intersection lengths

Let G be a graph and let $s \colon \mathrm{E}(G) \to \mathbb{N}^+$ prescribe the desired intersection lengths. In this section, finding a minimal s-respecting interval representation of G is reduced to finding (ℓ, s)-respecting interval representations. In particular, it is shown that the lengths of the intervals in a minimal s-respecting representation are determined by G and s, and can be computed efficiently.

Note that minimality is needed here, in contrast to the case of (ℓ, s)-respecting representations. The reason is that adding a point to an interval of an (ℓ, s)-respecting representation always destroys this property, while in an s-respecting representation, points that are contained in only one interval can be duplicated.

Lemma 4.5.6. *Let G be an interval graph with the length function $s \colon \mathrm{E}(G) \to \mathbb{N}^+$, and let $\alpha \colon \mathrm{V}(G) \to \mathcal{I}$ be any minimal s-respecting interval representation of G. Then the interval lengths $\ell(v) = |\alpha(v)|$ do not depend on the choice of α and can be computed from G and s in logspace; or in $O(n + m)$ time, if R_s is given as additional input.*

Proof. The algorithm works as follows. For each $v \in \mathrm{V}(G)$, consider these cases:

1. If $\mathrm{N}(v) = \emptyset$, set $\ell(v) = 1$.
2. If $\exists w \in \mathrm{N}(v) : v \subseteq_s w$, then set $\ell(v) = s(v, w)$.
3. Else, if there are $w_1, w_2 \in \mathrm{N}(v)$ that overlap each other, and overlap v from different sides (i.e., if $v \between_s w_1 \between_s w_2 \between_s v$ and $s(w_1, w_2) < \min\{s(w_1, v), s(w_2, v)\}$, cf. Lemma 4.5.3), then set $\ell(v) = s(w_1, v) + s(w_2, v) - s(w_1, w_2)$.
4. Otherwise, consider the subgraph $G[\mathrm{N}(v)]$ and define $\ell_v \colon \mathrm{N}(v) \to \mathbb{N}^+$ by $\ell_v(w) = s(w, v)$ for all $w \in \mathrm{N}(v)$. Additionally, define $s_v \colon \big(\mathrm{E}(G) \cap \binom{\mathrm{N}(v)}{2}\big) \to \mathbb{N}^+$ by $s_v(w_1, w_2) = \min\{s(w_1, v), s(w_2, v)\}$ if w_1 and w_2 overlap v from the same side (see Lemma 4.5.3), and $s_v(w_1, w_2) = s(w_1, w_2)$ otherwise. Compute an (ℓ_v, s_v)-respecting interval representation $\alpha_v \colon \mathrm{N}(v) \to \mathcal{I}_v$ of $G[\mathrm{N}(v)]$, and set $\ell(v) = |\bigcup_{I \in \mathcal{I}_v} I|$.

It needs to be shown that the computed ℓ satisfies $\ell(v) = |\alpha(v)|$ for each $v \in \mathrm{V}(G)$. For an isolated vertex v, as considered in case 1, the minimality of α implies $|\alpha(v)| = 1$, so $\ell(v) = 1$ is correct. By Lemma 4.5.1(b) and the definitions of $\between_s$ and $\subseteq_s$, it holds that $u \between_s v \Leftrightarrow \alpha(u) \between \alpha(v)$ and $u \subseteq_s v \Leftrightarrow \alpha(u) \subseteq \alpha(v)$. In case 2, this immediately implies $\ell(v) = s(v, w) = |\alpha(v) \cap \alpha(w)| = |\alpha(v)|$.

In case 3, $\alpha(w_1)$ and $\alpha(w_2)$ cover $\alpha(v)$, overlapping it from different sides (the latter is true by Lemma 4.5.3), giving rise to the situation depicted in Figure 4.10. Thus,

$$\begin{aligned}
\ell(v) &= s(w_1, v) + s(w_2, v) - s(w_1, w_2) \\
&= \big|\alpha(w_1) \cap \alpha(v)\big| + \big|\alpha(w_2) \cap \alpha(v)\big| - \big|\alpha(w_1) \cap \alpha(w_2)\big| \\
&= \big|\big(\alpha(w_1) \cup \alpha(w_2)\big) \cap \alpha(v)\big| = \big|\alpha(v)\big|.
\end{aligned}$$

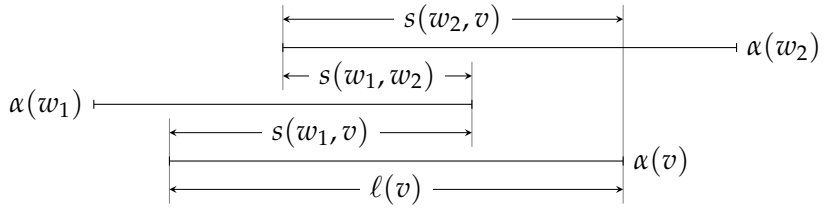

Figure 4.10: Proof of Lemma 4.5.6, case 3: $\alpha(w_1)$ and $\alpha(w_2)$ cover $\alpha(v)$, overlapping it from different sides.

In case 4, the definitions of ℓ_v and s_v truncate the intervals of the vertices in $\mathrm{N}(v)$ to include only their intersections with $\alpha(v)$. In particular, $|\alpha_v(u)| = |\alpha(u)|$ for all $u \subseteq_s v$, and $|\alpha_v(w)| = |\alpha(w) \cap \alpha(v)|$ for all $w \between_s v$. So truncating $\alpha\big(G[\mathrm{N}(v)]\big)$ gives an (ℓ_v, s_v)-respecting model $\alpha_v\big(G[\mathrm{N}(v)]\big)$ of $G[\mathrm{N}(v)]$. By Theorem 4.5.5, this model is unique up to isomorphism; in particular, its length is uniquely determined, implying $|\alpha(v)| \geq \ell(v)$. Indeed, both values are equal, for if $|\alpha(v)| > \ell(v)$ then the part of $\alpha(G)$ that is covered by $\alpha(v)$ could be replaced by the shorter $\alpha_v\big(G[\mathrm{N}(v)]\big)$, contradicting the minimality of α.

It is obvious that this algorithm can be implemented in logspace. To see that it is also possible in linear time, observe that in case 3, Lemma 4.5.3 allows to partition the $\between_s$-neighbors of v into two sets W_1 and W_2 so that neighbors that overlap from the same

side are in the same set. When choosing $w_i \in \mathrm{N}(v)$, for $i \in \{1,2\}$, it thus can be assumed that $w_i \in W_i$ and moreover that $s(w_i, v) = \max\{s(w, v) \mid w \in W_i\}$.

For the linear-time implementation of case 4, observe that each vertex u of G can occur in at most three of the auxiliary graphs: Suppose to the contrary that there are vertices v_1, v_2, v_3, v_4 such that $u \in \mathrm{N}(v_i)$ for each $i \in [1,4]$, and that case 4 is reached for each of these vertices. As case 2 does not apply, there are no containments, so we can assume $v_1^- < v_2^- < v_3^- < v_4^-$ and $v_1^+ < v_2^+ < v_3^+ < v_4^+$. As case 3 applies neither, it follows that $v_1^+ < v_3^-$ and $v_2^+ < v_4^-$. Now let $\alpha(u) = [u^-, u^+]$. As u is a neighbor of all v_i, we know $u^- \leq v_1^+$ and $v_4^- \leq u^+$. But this implies that $\alpha(u)$ either covers $\alpha(v_2)$ alone or together with $\alpha(v_1)$, contradicting that case 4 is reached for v_2. Note that case 3 is subsumed by case 4 and is only necessary to obtain the linear time bound. □

The following is a consequence of Theorem 4.5.5 and Lemmas 4.5.2 and 4.5.6.

Corollary 4.5.7. *Given a graph G and $s\colon \mathrm{E}(G) \to \mathbb{N}^+$, it is possible in $O(nm)$ time (resp., in logspace) to compute a minimal s-respecting interval representation $\alpha\colon \mathrm{V}(G) \to \mathcal{I}$ of G, or to detect that none exists. Moreover, for any minimal s-respecting interval representation $\alpha'\colon \mathrm{V}(G) \to \mathcal{I}'$ of G, there is a hypergraph isomorphism φ from $\mathcal{I}$ to $\mathcal{I}'$ with $\varphi\big(\alpha(v)\big) = \alpha'(v)$ for all $v \in \mathrm{V}(G)$.*

4.5.4 Interval graphs with unique maxclique ordering

As mentioned before, deciding if a graph has an ℓ-respecting interval representation is NP-complete [PS97]. This changes however, if the input graph G is required to have a unique interval ordering of its maxcliques (up to reflection); such a graph is called *UCO* for unique clique order. Equivalently, a graph G is UCO if and only if its bundle hypergraph $\mathcal{B}(G)$ admits a unique interval representation (up to reflection). On UCO graphs, even the more general problem DCIG (short for *distance constrained interval graph*) becomes tractable: Additionally to G, a system D of difference inequalities of the form $x_i - x_j \leq c$ is given, where the variables are the extreme points of the intervals (strict inequalities are allowed, too). The problem is to decide if G has an interval model that satisfies these inequalities. Pe'er and Shamir show that DCIG for UCO graphs is linear-time equivalent to the problem NegCycle, i.e., deciding if a weighted digraph has a negative cycle [PS97]. Based on the following facts, it follows that this problem is NL-complete.

Fact 4.5.8. NegCycle *with polynomially bounded weights is* NL*-complete.*

Proof. To check if a graph G has a negative cycle, nondeterministically walk through G (along edges) for at most $|V|$ steps, i.e., guess a vertex $v_0 \in \mathrm{V}(G)$, and proceed from v_i by guessing $v_{i+1} \in \mathrm{N}^+(v_i) = \{w \in \mathrm{V}(G) \mid (v_i, w) \in \mathrm{E}(G)\}$. When $v_{i+1} = v_0$ holds, the walk is not extended further. During the walk, store the start vertex v_0, the current vertex v_i, the number i of steps taken so far, and the accumulated weight of the traversed edges; this can be done in logarithmic space. Accept if the guessed walk returns to the start vertex and has negative weight, otherwise reject. If G contains a negative cycle $v_0, \ldots, v_k, v_0$, the computation that follows this cycle accepts. On the other hand, any accepting computation witnesses a negative closed walk, which must contain a negative cycle. Thus NegCycle is in NL.

To show the hardness, consider the following reduction from the NL-complete problem s-t-Con to decide if there is a directed path from s to t in a given digraph. An instance $\langle G, s, t\rangle$ of s-t-Con is reduced to the NegCycle instance $\langle G', w\rangle$, where the digraph G' is obtained from G by introducing the edge (t, s) if it is not yet present, and the weight function $w\colon \mathrm{E}(G') \to \mathbb{Z}$ is given by $w(s, t) = -|\mathrm{V}(G)|$ and $w(u, v) = 1$ for all other edges $(u, v) \in \mathrm{E}(G') \setminus \{(s, t)\}$. If there is a path from s to t in G, it has length (and thus weight) at most $|\mathrm{V}(G)| - 1$. Combining this path with the edge (t, s) results in a negative cycle in G'. Conversely, if there is a negative cycle in G', it must contain the edge (t, s), as this is the only one with negative weight. Cutting (t, s) out of that cycle results in a path from s to t in G. □

As an intermediate step in their reductions, Pe'er and Shamir consider the problem DiffIneq [PS97], which asks whether a system of difference inequalities admits an integral solution, i.e., the inequalities have the form $x_i - x_j \leq c_k$ or $x_i - x_j < c_k$, and the constants on the right hand side are polynomially bounded.

Fact 4.5.9. DiffIneq *is* NL*-complete.*

Proof. Pe'er and Shamir prove that the general DiffIneq problem reduces to its special case where all inequalities are weak [PS97, Lemma 3.2], by choosing $\epsilon < \frac{1}{n}$ (where n is the number of variables), replacing each strict inequality $x - y < c$ by $x - y \leq c - \epsilon$, and scaling the constants by $\Theta(n)$ to restore integrality. Thus we can assume that all inequalities are weak.

As NL is closed under complementation [Imm88; Sze88], Fact 4.5.8 implies that the problem NoNegCycle of deciding whether a given weighted digraph does *not* contain a negative cycle is NL-complete as well. It is not hard to see that a NoNegCycle instance $\langle G, w\rangle$ is equivalent to the DiffIneq instance $\langle X, A\rangle$ with variables $X = \mathrm{V}(G)$ and difference inequalities $A = \{x_i - x_j \leq w(x_j, x_i) \mid (x_j, x_i) \in \mathrm{E}(G)\}$ [see e.g. AMO93, pp. 103 sqq.]. Note that this construction can easily be reversed to obtain a reduction in the opposite direction.

For each walk $(v_1, \ldots, v_k)$ of weight $W = \sum_{i=1}^{k-1} w(v_i, v_{i+1})$ in $\langle G, w\rangle$, any assignment $\sigma\colon X \to \mathbb{Z}$ that satisfies all difference inequalities in A also satisfies $\sigma(v_k) \leq \sigma(v_1) + W$; this can be shown by an easy induction on k. If $\langle G, w\rangle$ contains a negative cycle $(v_1, \ldots, v_{k-1}, v_1)$, this implies $\sigma(v_1) < \sigma(v_1)$, contradicting the existence of a valid assignment.

On the other hand, if $\langle G, w\rangle$ does not have a negative cycle, fix an ordering $v_1, \ldots, v_n$ of $\mathrm{V}(G)$ such that whenever there is a directed path from v_j to v_i for $i < j$, then there is also a directed path from v_i to v_j. Such an ordering can be obtained by finding the strongly connected components, contracting each of them to a single vertex, taking a topological ordering of the resulting acyclic digraph, and substituting each representative vertex with the vertices of the strongly connected component it stands for. To construct a valid assignment $\sigma\colon X \to \mathbb{Z}$, let $\sigma(v_1) = 0$, and for $j > 1$ let $\sigma(v_j) = \min\{\sigma(v_i) + w(v_i, v_j) \mid v_i \in \mathrm{N}^-(v_j) \wedge i < j\}$, where $\mathrm{N}^-(v) = \{v \in \mathrm{V}(G) \mid (u, v) \in \mathrm{E}(G)\}$. This immediately satisfies all difference inequalities in A that correspond to edges of G that go forward w.r.t. the chosen ordering of $\mathrm{V}(G)$, and also the others as there are no negative cycles. □

Together with Fact 4.5.9, the next lemma implies that DCIG for UCO graphs is in NL.

Lemma 4.5.10. *The disjunctive reduction from DCIG for UCO graphs to* DiffIneq *given by Pe'er and Shamir [PS97] can be implemented in logspace.*

Proof. Let G be the input graph, and let D be the distance constraints for the extreme points of the intervals. The idea for the reduction is to extend D with additional constraints to two systems A and A' of difference inequalities, so that any interval representation of G that satisfies D corresponds to a satisfying assignment for one of A and A', and vice versa.

Any interval representation ρ of the bundle hypergraph $\mathcal{B}(G)$ induces the partial order $\prec_\rho$ on $\mathrm{V}(G)$ that has $u \prec_\rho v$ if and only if $\rho(\mathrm{B}_u)$ lies completely left of $\rho(\mathrm{B}_v)$. For $v \in \mathrm{V}(G)$, let v^- and v^+ denote the variables for its start and end point, respectively. Both A and A' contain all distance constraints in D. Additionally, for each pair of adjacent vertices if u and v, both systems contain the constraints $u^- < v^+$ and $v^- < u^+$ to ensure that their intervals intersect. For $u \prec_\rho v$, the inequality $u^+ < v^-$ is added to A, and the inequality $v^+ < u^-$ is added to A'. By assumption, ρ is unique up to reversal, so the result is unique up to exchanging A and A'.

Clearly, any solution to A (or to A') also satisfies D and specifies the extreme points of an interval representation of G. Conversely, in any interval representation α of G that satisfies D, the maxcliques occur in the order specified by either ρ or ρ^{-1}, and thus α satisfies one of A and A' [PS97, Lemma 3.1]. It remains to observe that ρ can be computed in logspace by Theorem 4.2.6; the rest of the reduction is easily possible in logspace. □

It remains to show that DCIG for UCO graphs is NL-hard.

Lemma 4.5.11. *The reduction from* DiffIneq *to the special case of DCIG on UCO graphs where all constraints are on the length of intervals [PS97, Section 3.2] can be implemented in logspace.*

Proof. The reduction works as follows. Let A be a system of difference inequalities on the variables $X = \{x_1, \ldots, x_n\}$. As argued in the proof of Fact 4.5.9, it can be assumed that all inequalities are weak, so write $A = \{x_{j_i} - x_{k_i} \leq c_i \mid i = 1, \ldots, n\}$. Fix $C > 1 + \sum_{i=1}^{m} |c_i|$ and let $c'_i = c_i + (j_i - k_i)C$.

The reduction maps A to the DCIG instance $\langle G, D \rangle$, where G is the intersection graph of the interval system $\mathcal{I}_1 \cup \mathcal{I}_2 \cup \mathcal{I}_3$, defined by

$$\begin{aligned}
\mathcal{I}_1 &= \{a_i \mid i = 0, \ldots, n\} && \text{where } a_i = [i, i+1] \\
\mathcal{I}_2 &= \{b_{i/2} \mid i = 0, \ldots, 2n+1\} && \text{where } b_{i/2} = [\tfrac{i}{2}, \tfrac{i}{2}] \\
\mathcal{I}_3 &= \{w_i \mid i = 0, \ldots, m\} && \text{where } w_i = [k_i, j_i] \text{ if } j_i < k_i \\
& && \quad \text{and } w_i = [j_i + \tfrac{1}{4}, k_i - \tfrac{1}{4}] \text{ otherwise.}
\end{aligned}$$

For integral i, the constraints $b_i^+ - b_i^- = 0$ and $b_{i+1/2}^+ - b_{i+1/2}^- = 1$ are included in D. For $j_i > k_i$, the constraint $w_i^+ - w_i^- \leq c'_i$ is added to D; for $j_i < k_i$, the constraint $w_i^+ - w_i^- \geq -c'_i - \epsilon$ with $\epsilon < 1/n$ is added to D.

All these constructions are easily possible in logspace. □

Combining Theorem 4.5.5 and Corollary 4.5.7 with the algorithm of Theorem 4.2.6 that computes canonical interval representations of interval hypergraphs allows to compute *canonical* (ℓ, s)- and s-respecting interval representations in logspace. The same can also be done in linear time using e.g. the PQ-tree algorithms of Booth and Lueker [BL76].

Using the fact that FL is closed under composition, it is easy to generalize the logspace results of this section to the case where the prescribed lengths are rational: Bring all lengths to a common denominator and use the resulting numerators. This transformation is possible in logspace as iterated integer multiplication is in DLogTime-uniform TC^0 [HAB02].

The bottleneck in the $O(nm)$ time algorithm for computing s-respecting interval representations given by Corollary 4.5.7 is the enumeration of R_s (see Lemma 4.5.2). Can this also be implemented in linear or at least in $O(n^2)$ time?

Does the complexity of computing (ℓ, s)- and s-respecting interval representations increase, if the interval and intersection lengths are restricted only for some vertices? The techniques of this section are not directly applicable in this case, as the algorithm of Lemma 4.5.4 relies on the uniqueness of the representation, which is not necessarily preserved in the modified scenario.

4.6 Constrained intersection structure

Another way to constrain an interval representation of an interval graph is to prescribe the intersection structure between all pairs of intervals. This can be done in form of an *interval matrix* $\mu = (\mu_{u,v})_{u \neq v \in V}$ with entries $\mu_{u,v} \in \{\mathtt{di}, \mathtt{ov}, \mathtt{cd}, \mathtt{cs}\}$. A function $\alpha \colon V \to \mathcal{I}$, where $\mathcal{I}$ is an interval system, is an *interval representation* of μ if it holds for all $u \neq v \in V$ that

$$\begin{aligned} \alpha(u) \cap \alpha(v) = \emptyset &\Leftrightarrow \mu_{u,v} = \mathtt{di}, \\ \alpha(u) \between \alpha(v) &\Leftrightarrow \mu_{u,v} = \mathtt{ov}, \\ \alpha(u) \subsetneq \alpha(v) &\Leftrightarrow \mu_{u,v} = \mathtt{cd}, \text{ and} \\ \alpha(u) \supsetneq \alpha(v) &\Leftrightarrow \mu_{u,v} = \mathtt{cs}. \end{aligned}$$

Note that μ specifies an *underlying graph* G_μ on the vertex set $\mathrm{V}(G_\mu) = V$ that contains the edges $\mathrm{E}(G_\mu) = \left\{\{u, v\} \in \binom{V}{2} \mid \mu_{u,v} \neq \mathtt{di}\right\}$. Any interval representation of μ is also one of G_μ; on the other hand, an interval representation of G_μ is only one of μ if it additionally satisfies the restrictions given by the non-di entries of μ.

Let $O = (O_u)_{u \in V}$ be a family of equivalence relations such that each O_u partitions the set $\{v \in V \mid \mu_{u,v} = \mathtt{ov}\}$ of ov-neighbors of u into at most two equivalence classes. An interval representation α of μ is called *O-respecting* if for any three vertices $u, v, w \in V$ with $\mu_{u,v} = \mu_{u,w} = \mathtt{ov}$, the intervals $\alpha(v)$ and $\alpha(w)$ contain the same extreme point of $\alpha(u)$ if and only if $(v, w) \in O_u$.

This section gives a logspace reduction from computing O-respecting interval representations of interval matrices to computing (ℓ, s)-respecting interval representations of interval graphs.

As a first step, the following lemma observes that an (O-respecting) interval representation α of an interval matrix μ can be required to be sharp, i.e., that every point of the resulting interval system $\alpha(\mu)$ is the extreme point of exactly one interval.

Lemma 4.6.1. *Let $\mu = (\mu_{u,v})_{u \neq v \in V}$ be an interval matrix and let $O = (O_u)_{u \in V}$ be a family of equivalence relations. Given any interval representation $\alpha \colon V \to \mathcal{I}$ of μ, a sharp interval representation $\alpha' \colon V \to \mathcal{I}'$ of μ can be obtained in logspace. Moreover, if α is O-respecting, then so is α'.*

Proof. Intuitively, the sharp interval representation α' is obtained from α by dropping all points that are not an extreme point of an interval and by individualizing coinciding extreme points as outlined in Figure 4.11: Place the start points in descending order of interval length, followed by the end points in ascending order of interval length. Note that these modifications preserve overlaps and inclusions, so α' is an (O-respecting) interval representation of μ.

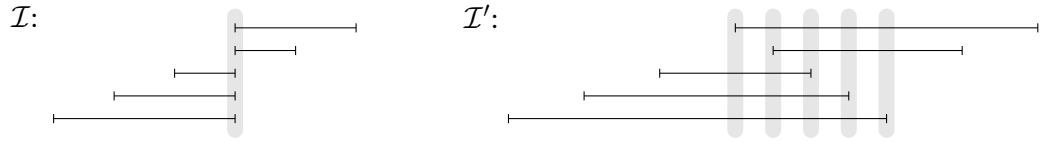

Figure 4.11: Proof of Lemma 4.6.1: How to order coinciding extreme points.

Formally, α' is defined as follows. For each $v \in V$, denote $\alpha(v) = [v^-, v^+]$ and let $\alpha'(v) = \big[l(v), r(v)\big]$, where

$$l(v) = \big|\{w \in V \,\big|\, w^- < v^-\}\big| + \big|\{w \in V \,\big|\, w^+ < v^-\}\big| + \big|\{w \in V \,\big|\, w^- = v^- \wedge w^+ > v^+\}\big|,$$
$$r(v) = \big|\{w \in V \,\big|\, w^- \leq v^+\}\big| + \big|\{w \in V \,\big|\, w^+ < v^+\}\big| + \big|\{w \in V \,\big|\, w^+ = v^+ \wedge w^- > v^-\}\big|.$$

Clearly, α' can be computed in logspace. The following equivalences follow directly from the definition of α'.

$$\begin{aligned}
l(u) < l(v) &\Leftrightarrow u^- < v^- \vee \big(u^- = v^- \wedge u^+ > v^+\big)\\
r(u) < r(v) &\Leftrightarrow u^+ < v^+ \vee \big(u^+ = v^+ \wedge u^- > v^-\big)\\
l(u) < r(v) &\Leftrightarrow u^- \leq v^+\\
l(u) > r(v) &\Leftrightarrow u^- > v^+
\end{aligned}$$

These equivalences imply $\alpha(u) \subsetneq \alpha(v) \Leftrightarrow \alpha'(u) \subsetneq \alpha'(v)$ and $\alpha(u) \between \alpha(v) \Leftrightarrow \alpha'(u) \between \alpha'(v)$, so α' is like α an interval representation of μ. Moreover, if α is O-respecting, then so is α'. To see that α' is sharp, observe that for all $v \in V$, it holds that $l(v), r(v) \in \{0, \ldots, 2 \cdot |V| - 1\}$, and the above equivalences imply $l(u) \neq r(v)$ for all $u, v \in V$ as well as $l(u) \neq l(v)$ and $r(u) \neq r(v)$ for all $u \neq v \in V$. (Note that $u \neq v$ implies $\alpha(u) \neq \alpha(v)$, as all inclusions allowed by μ are proper.) □

The key observation is that the entries of an interval matrix $\mu = (\mu_{u,v})_{u \neq v \in V}$ together with the equivalence relations in $O = (O_u)_{u \in V}$ already determine the lengths of all intervals and their pairwise intersections in any sharp interval representation of μ. Let G_μ

be the underlying graph. Define the functions $\ell_\mu \colon V \to \mathbb{N}^+$ and $s_{\mu,O} \colon \mathrm{E}(G_\mu) \to \mathbb{N}^+$ as follows:

$$\ell_\mu(v) = 2 + 2 \cdot \left|\left\{w \in V \setminus \{v\} \mid \mu_{v,w} = \mathtt{cs}\right\}\right| + \left|\left\{w \in V \setminus \{v\} \mid \mu_{v,w} = \mathtt{ov}\right\}\right|$$

$$s_{\mu,O}(\{u,v\}) = \begin{cases} \ell_\mu(u) & \text{if } \mu_{u,v} = \mathtt{cd} \\ \ell_\mu(v) & \text{if } \mu_{u,v} = \mathtt{cs} \\ 2 + 2 \cdot \left|\left\{w \in V \setminus \{u,v\} \mid \mu_{u,w} = \mathtt{cs} \wedge \mu_{v,w} = \mathtt{cs}\right\}\right| & \\ \quad + \left|\left\{w \in V \setminus \{u,v\} \mid \mu_{u,w} = \mathtt{ov} \wedge \mu_{v,w} = \mathtt{cs}\right\}\right| & \\ \quad + \left|\left\{w \in V \setminus \{u,v\} \mid \mu_{u,w} = \mathtt{cs} \wedge \mu_{v,w} = \mathtt{ov}\right\}\right| & \\ \quad + \left|\left\{w \in V \setminus \{u,v\} \mid \mu_{u,w} = \mu_{v,w} = \mathtt{ov} \wedge (u,v) \in O_w\right\}\right| & \text{if } \mu_{u,v} = \mathtt{ov} \end{cases}$$

The next lemma shows that ℓ_μ and $s_{\mu,O}$ reflect the lengths of the intervals and pairwise intersections in any sharp O-respecting interval representation of μ.

Lemma 4.6.2. *Let $\mu = (\mu_{u,v})_{u \neq v \in V}$ be an interval matrix and let $O = (O_u)_{u \in V}$ be a family of equivalence relations. Then any sharp O-respecting interval representation $\alpha \colon V \to \mathcal{I}$ of μ is an $(\ell_\mu, s_{\mu,O})$-respecting interval representation of the underlying graph G_μ.*

Proof. Let α be a sharp O-respecting interval representation of μ. This directly implies that α is also an interval representation of the underlying graph G_μ.

To see that α is ℓ_μ-respecting, consider any $v \in V$. As α is sharp, the cardinality of the interval $\alpha(v)$ is exactly the number of extreme points contained in it. Clearly, $\alpha(v)$ must contain both of its own extreme points. For each other vertex $w \in V \setminus \{v\}$, the matrix entry $\mu_{v,w}$ determines how many extreme points of $\alpha(w)$ are in $\alpha(v)$. This is illustrated in Figure 4.12(a). Indeed, if $\mu_{v,w} \in \{\mathtt{di}, \mathtt{cd}\}$ then no extreme point of $\alpha(w)$ is in $\alpha(v)$, if $\mu_{v,w} = \mathtt{cs}$ then both extreme points of $\alpha(w)$ are in $\alpha(v)$, and if $\mu_{v,w} = \mathtt{ov}$ then exactly one extreme point of $\alpha(w)$ is in $\alpha(v)$. Thus $\left|\alpha(v)\right| = \ell_\mu(v)$.

To see that α is $s_{\mu,O}$-respecting, consider any $\{u,v\} \in \mathrm{E}(G_\mu)$. Again, the entries of μ, this time together with the equivalence relations in O, determine the cardinality of $\alpha(u) \cap \alpha(v)$. If $\mu_{u,v} \in \{\mathtt{cd}, \mathtt{cs}\}$, then the length of the contained interval equals the length of the intersection. The case $\mu_{u,v} = \mathtt{ov}$ is more complicated and illustrated in Figure 4.12(b). The intersection $\alpha(u) \cap \alpha(v)$ contains exactly one extreme point for each of $\alpha(u)$ and $\alpha(v)$. The contribution of every other vertex $w \in V \setminus \{u,v\}$ is determined by the matrix entries $\mu_{u,w}$ and $\mu_{v,w}$ and the equivalence relation O_w. Indeed, if at least one of these entries is $\mathtt{di}$ or $\mathtt{cd}$, then no extreme point of $\alpha(w)$ lies in $\alpha(u) \cap \alpha(v)$; if both entries are $\mathtt{cs}$, then both extreme points of $\alpha(w)$ lie in $\alpha(u) \cap \alpha(v)$; and if one entry is $\mathtt{ov}$ and

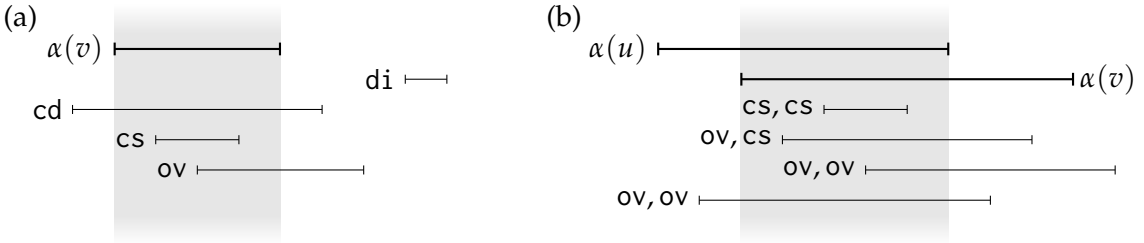

Figure 4.12: Proof of Lemma 4.6.2: The contribution of different types of neighbors to (a) $\ell_\mu(v)$ and (b) $s_{\mu,O}(\{u,v\})$ in case $\mu_{u,v} = \mathtt{ov}$.

the other is cs, then exactly one extreme point of $\alpha(w)$ lies in $\alpha(u) \cap \alpha(v)$. If both entries are ov, there are two cases: If $\alpha(w)$ overlaps both $\alpha(u)$ and $\alpha(v)$ from the same side, i.e. if $(u, v) \in O_w$, then one extreme point of $\alpha(w)$ lies in $\alpha(u) \cap \alpha(v)$, otherwise none. Thus $\big|\alpha(u) \cap \alpha(v)\big| = s_{\mu,O}\big(\{u, v\}\big)$. □

These two lemmas allow to compute interval representations of interval matrices.

Theorem 4.6.3. *Given an interval matrix $\mu = (\mu_{u,v})_{u \neq v \in V}$ and a family $O = (O_u)_{u \in V}$ of equivalence relations, a sharp O-respecting interval representation $\alpha \colon V \to \mathcal{I}$ of μ can be computed in logspace; if this is not possible, the algorithm detects this. Moreover, for any sharp O-respecting interval representation $\alpha' \colon V \to \mathcal{I}'$ of μ, there is a hypergraph isomorphism φ from $\mathcal{I}$ to $\mathcal{I}'$ with $\varphi\big(\alpha(v)\big) = \alpha'(v)$ for all $v \in V$.*

Proof. If μ admits a sharp O-respecting interval representation, Lemma 4.6.2 shows that the underlying graph G_μ admits an $(\ell_\mu, s_{\mu,O})$-respecting interval representation. Note that the graph G_μ and the functions ℓ_μ and $s_{\mu,O}$ can easily be computed from μ and O in logspace. By Theorem 4.5.5, an $(\ell_\mu, s_{\mu,O})$-respecting interval representation $\alpha_0 \colon V \to \mathcal{I}_0$ of G_μ can be computed in logspace; if the algorithm detects that there is no such α_0, the input μ and O is rejected. Moreover, the resulting interval model $\mathcal{I}_0$ is unique up to isomorphism. It is not hard to see that α_0 is also an interval representation of μ. Indeed, $\mu_{u,v} = \texttt{di}$ is equivalent to $\{u, v\} \notin \mathrm{E}(G_\mu)$ and thus to $\alpha_0(u) \cap \alpha_0(v) = \emptyset$; and likewise $\mu_{u,v} = \texttt{cd}$ is equivalent to $\ell_\mu(u) = s_{\mu,O}\big(\{u, v\}\big) < \ell_\mu(v)$ and thus to $\alpha_0(u) \subsetneq \alpha_0(v)$. Further, Lemma 4.5.3 implies that α_0 is O-respecting. Note though, that while the interval model $\mathcal{I}_0$ must be isomorphic to a sharp interval model of μ, it does not need to be sharp itself; for example, $\mathcal{I}_0$ might be $\big\{[1, 4], [3, 4]\big\}$. However, Lemma 4.6.1 allows to transform α_0 into a sharp interval representation $\alpha \colon V \to \mathcal{I}$ of μ in logspace. By Lemma 4.6.2, α is $(\ell_\mu, s_{\mu,O})$-respecting, so by the uniqueness part of Theorem 4.5.5 there is a hypergraph isomorphism φ from $\mathcal{I}$ to $\mathcal{I}_0$ with $\varphi(\alpha(v)) = \alpha_0(v)$ for all $v \in V$. Applying the same argument to an arbitrary sharp O-respecting interval representation $\alpha' \colon V \to \mathcal{I}'$ of μ proves the *moreover* part of the theorem. □

Combining this with the logspace algorithm of Theorem 4.2.6 for canonical representation of interval hypergraphs gives the following corollary.

Corollary 4.6.4. *Canonical O-respecting interval representations of interval matrices can be computed in logspace.*

4.7 Completeness results

Being able to compute canonical interval representations for a (hyper)graph class in FL immediately implies that the isomorphism problem of that class is in L. Thus the isomorphism problem of interval hypergraphs, interval graphs, and convex graphs is in L by Theorem 4.2.6, Corollary 4.3.1, and Corollary 4.3.2. Moreover, there is a standard Turing reduction of the automorphism group problem (i.e., computing a generating set of the automorphism group of a given graph) to the search version of the isomorphism problem for colored graphs [cf. Hof82; KST93]. It is not hard to see that this reduction can be performed in logspace.

Corollary 4.7.1. *Computing a generating set of the automorphism group of a given interval graph, and hence computing a canonical labeling coset for a given interval graph, is in* FL*. Further, the automorphism problem (i.e., deciding if a given graph has a nontrivial automorphism) for interval graphs is in* L*. The same holds for interval hypergraphs and convex graphs.*

In this section, it is shown that these problems are also hard for L.

Theorem 4.7.2. *The isomorphism and automorphism problems for interval graphs, bipartite permutation graphs, biconvex graphs, and convex graphs are* L*-complete under first-order translations.*

Bipartite permutation graphs are a subclass of biconvex graphs, which are in turn a subclass of convex graphs; so logspace algorithms for these classes are already presented in the previous sections. To prove hardness, the following lemma shows that the isomorphism and automorphism problems are L-hard even for caterpillars, a subclass of the mentioned graph classes. Caterpillars are trees that become paths when all leaves are removed.

Lemma 4.7.3. *The automorphism problem for caterpillars is* L*-hard under first-order translations.*

Proof. The hardness is shown using a reduction from PathCenter, which is L-hard by Lemma 2.8.1. Given an instance $\langle P, c\rangle$, we can assume that c has distance at least two to both ends; otherwise the instance can be trivially decided. The reduction constructs the graph G that is obtained from P by adding a new vertex c' and an edge $\{c, c'\}$; see Figure 4.13 for an example. It is clear that G is a caterpillar and that this reduction can be implemented as first-order translation. If c is the center of P, then G has the nontrivial automorphism that reflects P and maps c' to itself. If c is not the center of P, consider any automorphism φ of G. It must map c to itself as it is the only vertex of degree 3. Also, it must map c' and the ends of p to themselves, as they are the only vertices of degree 1 and all of them have a different distance to c. This implies that the other vertices are fixed as well, so φ must be the identity. □

The following lemma can be proved using a similar construction, which also marks either of the two end vertices of P (using two additional vertices – see Figure 4.14) in G_1 and G_2, respectively.

Lemma 4.7.4. *The isomorphism problem for caterpillars is* L*-hard under first-order translations.*

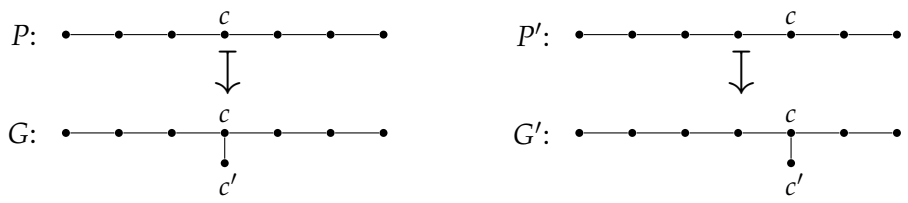

Figure 4.13: The reduction from PathCenter to the automorphism problem of caterpillars.

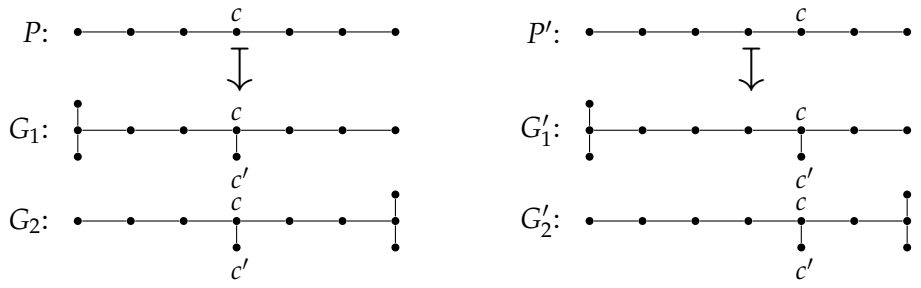

Figure 4.14: The reduction from PathCenter to the isomorphism problem of caterpillars.

These constructions can be modified to yield proper interval graphs by adding edges from the newly introduced marker vertices to all neighbors of the corresponding marked vertex (including other marker vertices).

Theorem 4.7.5. *The automorphism and isomorphism problems for proper interval graphs are* L*-complete under first-order translations.*

Another modification of these constructions can be used to show that the automorphism and isomorphism problems of interval hypergraphs are hard for L: Paths can also be viewed as 2-uniform hypergraphs (i.e., hypergraphs where all hyperedges have size 2). The only difference is that no new edges are added, but the hyperedges incident to the vertex that is to be marked are extended to also include the marker vertices.

Theorem 4.7.6. *The automorphism and isomorphism problems for interval hypergraphs are* L*-complete under first-order translations.*

The next topic of this section is the complexity of the recognition problems of the mentioned graph classes.

Theorem 4.7.7. *The recognition problem is* L*-complete under first-order translations for*

- *interval graphs,*
- *proper interval graphs,*
- *convex graphs,*
- *biconvex graphs,*
- *bipartite permutation graphs,*
- *caterpillars,*
- *paths, and*
- *interval hypergraphs, respectively.*

Proof. Recognition of interval graphs in logspace follows from the results of Reif [Rei84] and Reingold [Rei08] (or alternatively from Corollary 4.3.1). Each of the classes of convex, biconvex, and bipartite permutation graphs admits a characterization in terms of so-called asteroidal triples [see BLS99, Proposition 6.2.1; Tuc72, Theorem 6]. This characterization enables recognition of each of the three classes in logspace by a simple reduction to the connectivity problem. Interval hypergraphs are also recognizable in logspace by using a characterization by Duchet [Duc78] (or alternatively Theorem 4.2.6). Proper interval graphs can be recognized in logspace using the results from Section 4.4, and also by

the following fact: An interval graph is proper if and only if it has no induced copy of $K_{1,3}$ [Rob69]; this property can be tested in AC^0.

The hardness is shown via a reduction from the L-complete problem Ord [Ete97] that maps positive instances to paths (which are included in all graph classes listed above and also are 2-uniform interval hypergraphs) and that maps negative instances to graphs which are the disjoint union of a path and an circle of odd length at least 5 (and thus are not in any of the listed classes).

Let $\langle P, s, t\rangle$ be an instance of Ord; recall that this problem asks whether the vertex s occurs before the vertex t on the directed path P. It can be checked in AC^0 if t is among the first two vertices in the path. If so, output a trivial *yes*- or *no*-instance depending on whether s has been encountered first.

If t is not among the first 2 vertices of P, then construct an undirected graph G from P as follows: For each vertex $v \in V(P)$, insert a new vertex v' after v. Replace the incoming edge of s with (t', s) and replace the edge (t, t') with an edge that connects the first vertex in P and t. Finally, forget about the directions of the edges. This construction can clearly be implemented as first-order translation. Figure 4.15 shows an illustration. It is easy to verify that positive instances of Ord are mapped to paths, while the images of negative instances contain a chordless circle of odd length at least 5. □

Remark 4.7.8. Using Reingold's undirected graph reachability algorithm [Rei08], it can be shown that recognition of bipartite graphs is in L. Together with a result by Reif [Rei84], Reingold's algorithm also implies that recognition of chordal graphs can be done in logspace. Thus, the proof of Theorem 4.7.7 also implies L-completeness of the recognition problems for these two graph classes.

Corollary 4.7.9. *Given a graph G and length functions $\ell\colon V(G) \to \mathbb{N}^+$ and $s\colon E(G) \to \mathbb{N}^+$, it is L-complete under first-order translations to decide if G admits an (ℓ, s)-respecting or an s-respecting interval representation.*

Proof. Recall that the reduction from Ord to interval graph recognition given in the proof of Theorem 4.7.7 maps all positive instances to paths. So these graphs have (ℓ, s)- and s-respecting interval representations if we let $\ell(v) = 2$ and $s(e) = 1$ for all $v \in V(G)$ and $e \in E(G)$. Finally, both problems are in L by Theorem 4.5.5 and Corollary 4.5.7. □

Corollary 4.7.10. *Given a matrix $\mu = (\mu_{u,v})_{u \neq v \in V}$ with entries from $\{\mathtt{di}, \mathtt{cd}, \mathtt{cs}, \mathtt{ov}\}$, it is L-complete under first-order translations to decide if μ is an interval matrix.*

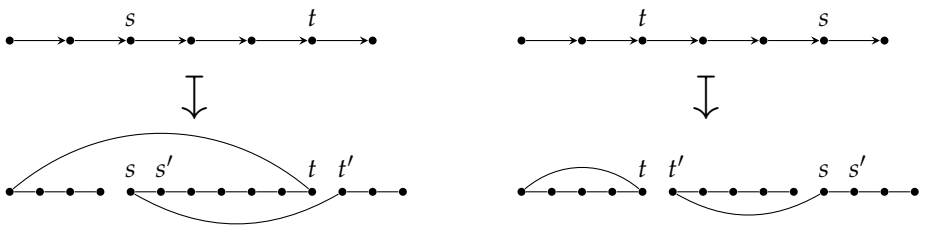

Figure 4.15: The reduction that maps positive instances of Ord to paths and negative instances to disjoint unions of a path and a circle of odd length at least 5.

Proof. Theorem 4.6.3 allows to find an interval representation of μ in logspace if one exists. The hardness follows from Theorem 4.7.7, because substituting di for 0 and ov for 1 in the adjacency matrix of a graph G results in an interval matrix if and only if G is a proper interval graph. □

Corollary 4.7.11. *The problems of computing a PEO and an interval representation for a given interval graph G are in* FL *and are hard for logspace.*

Proof. An interval representation can be constructed in logspace by Corollary 4.3.1. Any interval representation induces an ordering of $V(G)$; it is not hard to see that such an ordering is a PEO. On the other hand, in the proof of Theorem 3.4.3 we have seen that computing a PEO is hard for logspace even for paths by a reduction from Ord. □

5 Canonical representation of circular-arc graphs

The main results of this chapter are logspace algorithms for canonical arc representation of Helly circular-arc graphs, proper circular-arc graphs, and concave-round graphs.

At first sight, the circular-arc setting appears to be closely related to the interval setting. One might expect that the results from Chapter 4 can be easily transferred to the circular-arc world. Sometimes this is indeed possible, as with the canonical representation algorithm for CA hypergraphs presented in Section 5.1. It is based on Tucker's observation that any CA hypergraph becomes interval when all hyperedges that contain a fixed vertex are complemented [Tuc71].

However, the world of CA graphs is more complex. In Section 5.2, graphs that admit Helly arc models are considered. They are in some sense 'close' to interval graphs, because every interval model is Helly. But even HCA graphs require new techniques: As not every maxclique is the intersection of constantly many neighborhoods, the reduction to representation of the bundle hypergraph that was used in the interval case does not work. Instead, the logspace algorithm of Section 4.6 for finding interval representations with prescribed intersection structure is used as a subroutine.

Section 5.3 deals with canonical arc representation of proper circular-arc graphs and concave-round graphs. As in most recognition and isomorphism algorithms for these graph classes, two cases are distinguished: For any concave-round graph G with bipartite complement $\overline{G}$, it is shown that the open neighborhood hypergraph $\mathcal{N}(\overline{G})$ is interval; moreover, an interval representation of $\mathcal{N}(\overline{G})$ can be transformed in logspace into an arc representation of G, which can be made proper if G is PCA. For any concave-round graph G whose complement is non-bipartite, a tight arc representation of its neighborhood hypergraph $\mathcal{N}[G]$ can be transformed in logspace into a proper arc representation of G.

5.1 Canonical representation of circular-arc hypergraphs

Recall that a function $\mathcal{H} \mapsto \rho_{\mathcal{H}}$ defined on the class of CA hypergraphs computes canonical arc representations, if (a) $\rho_{\mathcal{H}}$ is an arc representation for any input hypergraph $\mathcal{H}$ and (b) isomorphic input hypergraphs $\mathcal{H} \cong \mathcal{K}$ result in equal arc models $\rho_{\mathcal{H}}(\mathcal{H}) = \rho_{\mathcal{K}}(\mathcal{K})$.

Theorem 5.1.1. *Canonical arc representations for CA hypergraphs can be computed in logspace.*

Proof. Let $\mathcal{H}$ be a CA hypergraph with n vertices. For a hyperedge $A \in \mathcal{H}$, let $m(A)$ denote its multiplicity. The complement of $A \in \mathcal{H}$ is $\overline{A} = \mathrm{V}(\mathcal{H}) \setminus A$.

For each vertex $x \in \mathrm{V}(\mathcal{H})$, construct the simple hypergraph

$$\mathcal{H}_x = \left\{ A \in \mathcal{H} \,\middle|\, x \notin A \right\} \cup \left\{ \overline{A} \,\middle|\, A \in \mathcal{H}, x \in A \right\}$$

on the same vertex set $\mathrm{V}(\mathcal{H}_x) = \mathrm{V}(\mathcal{H})$. Observe that every $\mathcal{H}_x$ is an interval hypergraph [cf. Tuc71, Theorem 1]. Define an edge-coloring c_x of $\mathcal{H}_x$ by $c_x(A) = \langle m(A), m(\overline{A})\rangle$, where $m(A) = 0$ for $A \notin \mathcal{H}$. These edge-colors can again be encoded as edge multiplicities $m_x(A) = p\big(c_x(A)\big)$ for $\mathcal{H}_x$ using the Cantor pairing function $p\colon \mathbb{N} \times \mathbb{N} \to \mathbb{N}$, which is the bijection defined by $p(i,j) = \frac{1}{2}(i+j)(i+j+1) + j$. Note that $p(i,j)$ is polynomial in i and j, and that $p\big(c_x(A)\big) > 0$ for all $A \in \mathcal{H}_x$ as $p(\langle 0,0\rangle) = 0$.

Canonizing each $\mathcal{H}_x$ (with edge multiplicities m_x) using the algorithm of Theorem 4.2.6 results in n interval representations $\rho_x\colon \mathrm{V}(\mathcal{H}) \to \{1,\dots,n\}$; recall that $\mathrm{V}(\mathcal{H}_x) = \mathrm{V}(\mathcal{H})$. Each ρ_x gives an arc model $\rho_x(\mathcal{H})$ of $\mathcal{H}$, which can be obtained from the corresponding canonical interval model $\rho_x(\mathcal{H}_x)$ of $\mathcal{H}_x$ as follows: For each hyperedge $A \in \mathcal{H}_x$ of multiplicity $m_x(A)$, let $\langle i,j\rangle = p^{-1}(m_x(A))$ and include i copies of $\rho_x(A)$ and j copies of the complement $\overline{\rho_x(A)}$. Among these n candidate arc models of $\mathcal{H}$, choose the lexicographically least arc model as canonical and output the corresponding arc representation ρ_x. □

In Corollary 4.2.7 it was observed that a logspace representation algorithm for interval hypergraphs can be used to check whether a matrix has the consecutive-ones property and to find an appropriate permutation of the columns. The logspace representation algorithm for CA hypergraphs of Theorem 5.1.1 allows the same for the circular-ones property.

Corollary 5.1.2. *There is a logspace algorithm that decides whether a given boolean matrix has the circular-ones property and computes an appropriate permutation of the columns if this is the case.*

The canonical representation algorithm given by Theorem 5.1.1 also solves the canonical labeling problem for CA hypergraphs in logarithmic space. This algorithm can also be used to compute canonical labelings for the duals of CA hypergraphs.

Corollary 5.1.3. *Canonical labelings for hypergraphs whose duals are CA can be computed in logspace.*

Proof. Recall that the dual $\mathcal{H}^{\mathsf{D}}$ of a hypergraph $\mathcal{H}$ has the vertices $\mathrm{V}(\mathcal{H}^{\mathsf{D}}) = \mathcal{H}$ and the hyperedges $\mathcal{H}^{\mathsf{D}} = \big\{\!\!\big\{ v^* \bigm| v \in \mathrm{V}(\mathcal{H}) \big\}\!\!\big\}$, where $v^* = \big\{\!\!\big\{ A \in \mathcal{H} \mid v \in A \big\}\!\!\big\}$. The map $\varphi\colon v \mapsto v^*$ is an isomorphism from $\mathcal{H}$ to $(\mathcal{H}^{\mathsf{D}})^{\mathsf{D}}$. If $\mathcal{H}^{\mathsf{D}}$ is a CA hypergraph, this map can be combined with a canonical labeling λ of $\mathcal{H}^{\mathsf{D}}$ in order to obtain a canonical labeling $\hat{\lambda}$ of $\mathcal{H}$. More precisely, $\hat{\lambda}$ is obtained from the map $\lambda'(v) = \big\{\!\!\big\{ \lambda(A) \bigm| A \ni v \big\}\!\!\big\}$ by sorting the list $\big(\lambda'(v)\big)_{v \in \mathrm{V}(\mathcal{H})}$ according to the lexicographic order and letting $\hat{\lambda}(v) = i$ if $\lambda'(v)$ ends up at position i. □

In analogy to convex graphs, Liang and Blum [LB95] call a bipartite graph G with vertex classes U and V *circular-convex*, if the hypergraph $\mathcal{N}_U(G) = \big\{\!\!\big\} \mathrm{N}_G(v) \bigm| v \in V \big\}\!\!\big\}$ on the vertex set $\mathrm{V}\big(\mathcal{N}_U(G)\big) = U$ is CA.

Corollary 5.1.4. *Canonical labelings for circular-convex graphs can be computed in logspace.*

Proof. Let G be a connected circular-convex graph with vertex classes U and V. Apply the algorithm of Theorem 5.1.1 to $\mathcal{N}_U(G)$ and if also to $\mathcal{N}_V(G)$; at least one of those hypergraphs is CA. The resulting representation(s) can be used to construct a canonical labeling for G as in the approach of Corollary 4.3.2, where it is also described how disconnected graphs can be handled. □

In Section 4.7 it was observed that the isomorphism problems for interval hypergraphs and convex graphs are L-hard. As these are subclasses of CA hypergraphs and circular-convex graphs, respectively, this hardness also holds for the isomorphism problem of the latter classes.

5.2 Canonical representation of Helly circular-arc graphs

In this section, a logspace algorithm for canonical representation of HCA graphs is presented. It proceeds in several steps; see Figure 5.1.

The first step is based on Hsu's observation that the structure of certain CA graphs G allows to prescribe the intersection structure of each pair of arcs in an arc representation $\mathcal{A}$ of G [Hsu95]. For each pair of arcs (A, B), their relationship is prescribed as either disjoint ($A \cap B = \emptyset$, denoted `di`), contained ($A \subsetneq B$, denoted `cd`), containing ($A \supsetneq B$, denoted `cs`), circle cover ($A \rotatebox[origin=c]{90}{)\!(} B$ and A contains both extreme points of B, denoted `cc`) or strict overlap ($A \rotatebox[origin=c]{90}{)\!(} B$ and A contains only one extreme point of B, denoted `ov`). The desired intersection types are stored in the *neighborhood matrix* λ_G of G which has entries from $\{\texttt{di}, \texttt{cd}, \texttt{cs}, \texttt{cc}, \texttt{ov}\}$ (for details see Section 5.2.1).

The motivation for switching to the matrix λ_G is that flipping the arc of a vertex (i.e., replacing it with the arc that has the same extreme points but covers the opposite part of the circle) can be mimicked in λ_G by substituting some of its entries. This substitution is described in Section 5.2.2, where it is also shown how a subset $X \subseteq V(G)$ can be identified such that flipping the arcs of all vertices in X results in a matrix λ' that can be realized by an interval system. The set X is chosen to be a maxclique of G that is the common neighborhood of two vertices, and it is shown that every HCA graph contains at least one such clique. A sharp interval representation $\alpha_0 \colon V(G) \to \mathcal{I}_{\lambda'}$ of the interval matrix λ' will be computed in logspace using the algorithm of Theorem 4.6.3. To obtain an arc representation of λ_G (and thus of G), the intervals of $\mathcal{I}_{\lambda'}$ that correspond to flipped vertices will be flipped back. Additionally, it is shown in Section 5.2.3 that the resulting sharp HCA model $\mathcal{A}_{\lambda_G}$ of λ_G is unique up to isomorphism; thus the canonical representation algorithm for CA hypergraphs of Theorem 5.1.1 can be used to obtain canonical arc representations.

$$\begin{array}{rcccl}
\text{interval matrix} & \lambda' & \longrightarrow & \mathcal{I}_{\lambda'} & \text{interval model} \\
& \uparrow & & \downarrow & \\
\text{HCA matrix} & \lambda_G & & \mathcal{A}_{\lambda_G} & \text{HCA model} \\
& \uparrow & & \| & \\
\text{HCA graph} & G & & \mathcal{A}_G & \text{HCA model}
\end{array}$$

Figure 5.1: Overview of the canonical representation algorithm for HCA graphs.

5.2.1 From HCA graphs to HCA matrices

Let $\mu = (\mu_{i,j})_{i \neq j \in V}$ be a quadratic matrix. The elements of V are called the *vertices* of μ and it is assumed that V is linearly ordered. Another quadratic matrix $\lambda = (\lambda_{i,j})_{i \neq j \in U}$ is *isomorphic* to μ (written $\lambda \cong \mu$) if there is a bijection $\sigma \colon U \to V$ such that $\lambda_{i,j} = \mu_{\sigma(i),\sigma(j)}$

for all $i \neq j \in U$. Note that two graphs are isomorphic if and only if their adjacency matrices are isomorphic.

An *intersection matrix* is a matrix $\mu = (\mu_{u,v})_{u \neq v \in V}$ with entries $\mu_{u,v} \in \{\texttt{di}, \texttt{cs}, \texttt{cd}, \texttt{cc}, \texttt{ov}\}$ that satisfies (a) $\mu_{u,v} = \texttt{cd} \Leftrightarrow \mu_{v,u} = \texttt{cs}$ and (b) $\mu_{u,v} = \mu_{v,u}$ in all other cases. The idea is that intersection matrices describe the intersection structure between the arcs of an arc system $\mathcal{A}$. When two arcs A and B overlap (i.e., $A \between B$), it will be helpful to distinguish the case where A contains both extreme points of B, which is called *circle cover* and denoted by $A \circledcirc B$, from the case where A contains only one extreme point of B, which is called *strict overlap* and denoted by $A \curvearrowright B$.

Recall that an arc system $\mathcal{A}$ is *sharp* if every point on the circle $\mathrm{V}(\mathcal{A})$ is the extreme point of exactly one arc in $\mathcal{A}$. The following notation was introduced by Lin and Szwarcfiter.

Definition 5.2.1 [LS09]**.** Let $\mathcal{A}$ be an arc system with $\mathrm{V}(\mathcal{A}) \notin \mathcal{A}$ where each arc has multiplicity 1. The *intersection matrix of* $\mathcal{A}$ is $\mu_{\mathcal{A}} = (\mu_{A,B})_{A \neq B \in \mathcal{A}}$, where the entries are

$$\mu_{A,B} = \begin{cases} \texttt{di} & \text{if } A \cap B = \emptyset; \\ \texttt{cd} & \text{if } A \subsetneq B; \\ \texttt{cs} & \text{if } A \supsetneq B; \\ \texttt{cc} & \text{if } A \circledcirc B \text{ (i.e., if } A \between B \text{ and } A \text{ contains both extreme points of } B); \\ \texttt{ov} & \text{if } A \curvearrowright B \text{ (i.e., if } A \between B \text{ and } A \text{ contains only one extreme point of } B). \end{cases}$$

A matrix μ is a *(Helly) circular-arc matrix* if there is a (Helly) arc system $\mathcal{A}$ such that $\mu \cong \mu_{\mathcal{A}}$.

Note that a matrix μ is an interval matrix as defined in Section 4.6 if and only if there is an interval system $\mathcal{I}$ such that $\mu \cong \mu_{\mathcal{I}}$.

Definition 5.2.2 [LS09]**.** Given a graph G without twins and universal vertices, its *neighborhood matrix* $\lambda_G = (\lambda_{u,v})_{u \neq v \in \mathrm{V}(G)}$ is defined by the entries

$$\lambda_{u,v} = \begin{cases} \texttt{di} & \text{if } \{u,v\} \notin \mathrm{E}(G); \\ \texttt{cd} & \text{if } \mathrm{N}[u] \subsetneq \mathrm{N}[v]; \\ \texttt{cs} & \text{if } \mathrm{N}[u] \supsetneq \mathrm{N}[v]; \\ \texttt{cc} & \text{if } \mathrm{N}[u] \between \mathrm{N}[v], \mathrm{N}[u] \cup \mathrm{N}[v] = V, \\ & \quad \text{and } \forall w \in \mathrm{N}[u] \setminus \mathrm{N}[v] : \mathrm{N}[w] \subset \mathrm{N}[u], \\ & \quad \text{and } \forall w \in \mathrm{N}[v] \setminus \mathrm{N}[u] : \mathrm{N}[w] \subset \mathrm{N}[v]; \\ \texttt{ov} & \text{otherwise.} \end{cases}$$

Note that λ_G can be viewed as an enriched adjacency matrix: Pairs of nonadjacent vertices are marked with $\texttt{di}$ and the entries for adjacent pairs are differentiated into the four other categories. The *underlying graph* of an intersection matrix $\mu = (\mu_{u,v})_{u \neq v \in V}$ is denoted by G_μ; it consists of the vertices V and the edges $\big\{\{u,v\} \bigm| \mu_{u,v} \neq \texttt{di}\big\}$.

An arc representation $\alpha \colon \mathrm{V}(G) \to \mathcal{A}$ of a graph G is *normalized* if $\mathcal{A}$ is sharp and α is an isomorphism between the neighborhood matrix λ_G and the intersection matrix $\mu_{\mathcal{A}}$, i.e., $\lambda_{u,v} = \mu_{\alpha(u),\alpha(v)}$ for all $u, v \in \mathrm{V}(G)$. Normalized arc representations were introduced by Hsu, who provides a linear-time algorithm that transforms any arc representation of a CA graph with certain properties into a normalized representation.

Lemma 5.2.3 [Hsu95, Corollary 2.3]**.** *Any CA graph G without twins and universal vertices has a normalized arc representation.*

Proof sketch. Let $\alpha\colon \mathrm{V}(G) \to \mathcal{A}$ be an arc representation of G. It can be assumed that $\mathcal{A}$ is sharp, as the proof of Lemma 4.6.1 can easily be adapted from the interval to the circular-arc setting.

Hsu's algorithm modifies α in two stages to ensure $\lambda_G = \mu_{\mathcal{A}}$. In the first stage, all arcs of $\mathcal{A}$ are extended as far as possible. This eliminates *type 1 violations*, where two vertices $u, v \in \mathrm{V}(G)$ exist with $\lambda_{u,v} = \mathsf{cc} \neq \mathsf{ov} = \mu_{\alpha(u),\alpha(v)}$. To be precise, the algorithm splits the circle into *blocks* of consecutive points, such that each block is a maximal sequence either of start points or of end points. For each start point of an arc $A = [a^-, a^+] \in \mathcal{A}$, the preceding block $E \subset \mathrm{V}(\mathcal{A})$ of end points is split into the set E_1 of end points whose arcs intersect with A and the remaining points E_2. Then the points in E are reordered so that those in E_1 come before those in E_2, and the start point a^- of A is moved between E_1 and E_2. Afterwards, the end point of each arc is treated symmetrically.

The second stage of the algorithm reorders the points within each block to get as many containments as possible. This resolves *type 2 violations*, where two vertices $u, v \in \mathrm{V}(G)$ exist with $\lambda_{u,v} = \mathsf{cd} \neq \mathsf{ov} = \mu_{\alpha(u),\alpha(v)}$. In each block of start points, the points are ordered by how far the corresponding arcs extend beyond this block, placing those first whose arcs extend furthest. After that, each block of end points is reordered by how far the corresponding arcs extend beyond this block, this time placing those points first whose arcs extend least.

It is not hard to see that this algorithm does not change the intersection graph of $\mathcal{A}$ and resolves all violations of $\lambda_G = \mu_{\mathcal{A}}$ [for details see Hsu95, p. 415]. □

All normalized arc representations have a property that is called *stable* by Joeris et al., who use their characterization of HCA graphs by forbidden induced subgraphs to prove that every stable arc representation of an HCA graph G yields an HCA model [JLM+11, Theorem 4.1]. This implies the following.

Lemma 5.2.4. *Let G be an HCA graph without twins and universal vertices. For any normalized arc representation $\alpha\colon \mathrm{V}(G) \to \mathcal{A}$ of G, the resulting arc model $\mathcal{A}$ is Helly.*

An *arc representation of a CA matrix* $\lambda = (\lambda_{u,v})_{u \neq v \in V}$ is a bijection α from V to some arc system $\mathcal{A}$ that has satisfies the intersection structure prescribed by λ, i.e., α must be an isomorphism from λ to the intersection matrix $\mu_{\mathcal{A}}$.

Let $O = (O_u)_{u \in V}$ be a family of equivalence relations such that each O_u partitions $\{v \in V \mid \lambda_{u,v} = \mathsf{ov}\}$ into at most two equivalence classes. As in the interval case, an arc representation α of λ is called *O-respecting*, if for any three vertices $u, v, w \in V$ with $\lambda_{u,v} = \lambda_{u,w} = \mathsf{ov}$, the arcs $\alpha(v)$ and $\alpha(w)$ contain the same extreme point of $\alpha(u)$ if and only if $(v, w) \in O_u$.

For a given HCA graph G, the following two lemmas allow to compute a family $O_G = (O_u)_{u \in \mathrm{V}(G)}$ such that the neighborhood matrix λ_G of G admits an O_G-respecting arc representation.

Lemma 5.2.5. *Let $\alpha\colon \mathrm{V}(G) \to \mathcal{A}$ be a normalized HCA representation of a graph G. Also, let $u, v, w \in \mathrm{V}(G)$ with $\alpha(u) \frown \alpha(v)$, $\alpha(u) \frown \alpha(w)$ and $\alpha(v) \frown \alpha(w)$. Then it holds that*

$$\alpha(u) \cap \alpha(v) \subseteq \alpha(w) \Leftrightarrow \mathrm{N}[u] \cap \mathrm{N}[v] \subseteq \mathrm{N}[w]\,.$$

Proof. The forward direction follows from the Helly property and $\alpha(u) \cap \alpha(v) \neq \emptyset$. Indeed, let $x \in N[u] \cap N[v]$. Thus $\alpha(x)$ intersects both $\alpha(u)$ and $\alpha(v)$. By the Helly property, $\alpha(x)$ intersects even $\alpha(u) \cap \alpha(v)$. By assumption, the latter set is contained in $\alpha(w)$, implying $x \in N[w]$ as desired.

Now suppose that $\alpha(u) \cap \alpha(v) \not\subseteq \alpha(w)$. We may assume without loss of generality that $\alpha(u)$ contains the start point of $\alpha(v)$; this will be denoted by $\alpha(u) \prec \alpha(v)$. As the Helly property precludes the case $\alpha(v) \prec \alpha(w) \prec \alpha(u)$ (see Fig. 5.2(a)), it follows that either $\alpha(u) \prec \alpha(w)$ and $\alpha(v) \prec \alpha(w)$ or $\alpha(w) \prec \alpha(u)$ and $\alpha(w) \prec \alpha(v)$. Assume the former relation (see Fig. 5.2(b)); the latter case is symmetric. Because α is normalized and $\alpha(v) \not\subseteq \alpha(w)$, there exists a vertex $x \in N[v] \setminus N[w]$. Its arc $\alpha(x)$ must contain a point in $\alpha(v) \setminus \alpha(w)$, which is a subset of $\alpha(u)$. Thus x is also a neighbor of u and witnesses that $N[u] \cap N[v] \not\subseteq N[w]$. □

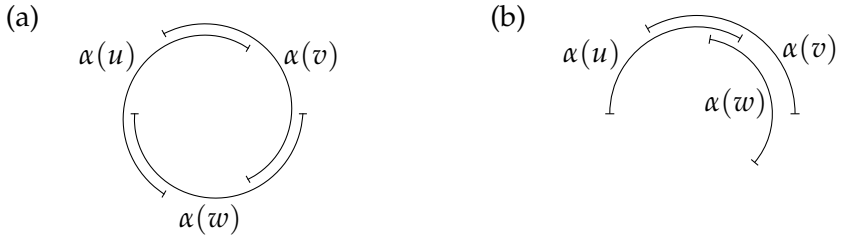

Figure 5.2: Proof of Lemma 5.2.5

For a graph G, define the family $O_G = (O_u)_{u \in V(G)}$, where O_u is an equivalence relation on $\{v \in V(G) \mid \lambda_{u,v} = \mathsf{ov}\}$, by

$$(v_1, v_2) \in O_u \Leftrightarrow v_1 = v_2 \vee \lambda_{v_1,v_2} \in \{\mathsf{cd}, \mathsf{cs}\} \vee \big(\lambda_{v_1,v_2} = \mathsf{ov} \wedge N[v_1] \cap N[v_2] \not\subseteq N[u]\big).$$

Lemma 5.2.6. *Let $\lambda_G = (\lambda_{u,v})_{u \neq v \in V(G)}$ be the neighborhood matrix of an HCA graph G without twins and universal vertices. Then any sharp arc representation $\alpha \colon V(G) \to \mathcal{A}$ of λ_G is O_G-respecting.*

Proof. Let $u, v_1, v_2 \in V(G)$ with $\lambda_{u,v_1} = \lambda_{u,v_2} = \mathsf{ov}$; it has to be shown that $(v_1, v_2) \in O_u$ if and only if $\alpha(v_1)$ and $\alpha(v_2)$ contain the same extreme point of $\alpha(u)$.

If $\lambda_{v_1,v_2} = \mathsf{cd}$ (and thus $(v_1, v_2) \in O_u$), it follows that $\alpha(v_1) \subsetneq \alpha(v_2)$. Thus the extreme point of $\alpha(u)$ contained in $\alpha(v_1)$ is also contained in $\alpha(v_2)$. The case $\lambda_{v_1,v_2} = \mathsf{cs}$ is symmetric.

In case $\lambda_{v_1,v_2} = \mathsf{di}$ (where $(v_1, v_2) \notin O_u$), it follows that $\alpha(v_1)$ and $\alpha(v_2)$ are disjoint and consequently cannot contain the same extreme point of $\alpha(u)$. Similarly, $\lambda_{v_1,v_2} = \mathsf{cc}$ (where $(v_1, v_2) \notin O_u$) implies $\alpha(v_1) \circledcirc \alpha(v_2)$, and as $\alpha(u)$ strictly overlaps both these arcs, its extreme points must lie in $\alpha(v_1) \setminus \alpha(v_2)$ and $\alpha(v_2) \setminus \alpha(v_1)$, respectively.

For the remaining case $\lambda_{v_1,v_2} = \mathsf{ov}$, the condition $(v_1, v_2) \in O_u$ holds if and only if $N[v_1] \cap N[v_2] \not\subseteq N[u]$. By Lemma 5.2.5, this is equivalent to $\alpha(v_1) \cap \alpha(v_2) \not\subseteq \alpha(u)$, which in turn holds exactly if $\alpha(v_1)$ and $\alpha(v_2)$ contain the same extreme point of $\alpha(u)$. □

Using these facts, the first step of the representation algorithm for HCA graphs can be described.

Lemma 5.2.7. *There is a logspace reduction from computing canonical HCA representations for HCA graphs to computing canonical sharp (O-respecting) arc representations of vertex-colored HCA matrices.*

Proof. First consider the case that the input graph G is twin-free and has no universal vertex. In this case, the reduction computes the neighborhood matrix λ_G (along with the family O_G), queries the oracle for a sharp (O_G-respecting) arc representation $\alpha_G \colon \mathrm{V}(G) \to \mathcal{A}_G$ of λ_G, and returns α_G as HCA representation of G. By definition, a function $\alpha \colon \mathrm{V}(G) \to \mathcal{A}$ is a normalized arc representation of G if and only if it is a sharp arc representation of λ_G. By Lemma 5.2.3, such an α exists, it is O_G-respecting by Lemma 5.2.6, and the resulting arc model $\mathcal{A}$ is Helly by Lemma 5.2.4. Regarding the canonicity, observe that $G \cong H$ is equivalent to $\lambda_G \cong \lambda_H$. Thus, if $\lambda_G \cong \lambda_H$ implies $\mathcal{A}_G = \mathcal{A}_H$, then so does $G \cong H$.

If the input graph G contains twins, apply the above algorithm to its quotient graph G', which has one vertex for each twin class $[v]$ of G and has the edge $\{[u],[v]\}$ whenever G has the edge $\{u,v\}$ (note that $\{u',v'\} \in \mathrm{E}(G)$ if and only if $\{u,v\} \in \mathrm{E}(G)$ for all $u' \in [u]$ and $v' \in [v]$, so $\mathrm{E}(G')$ is well-defined). Let $\alpha_{G'}$ be the computed HCA representation of G'. Then define a representation α of G by $\alpha(v) = \alpha'([v])$. To preserve canonicity, color each vertex $[v]$ of G' with the size of the twin class $[v]$; then $G \cong H$ is equivalent to $G' \cong H'$. It remains to observe that universal vertices can be removed before computing the neighborhood matrix, adding arcs for them afterwards. □

5.2.2 From HCA matrices to interval matrices

This section describes a logspace reduction from computing sharp O-respecting arc representation for HCA matrices to computing sharp O-respecting interval representations of interval matrices.

Given an arc $B = [b^-, b^+]$ on a circle $\mathbb{C}$, *flipping* B results in the arc $B^{\mathsf{F}} = [b^+, b^-]$ that has the same extreme points as B but covers the opposite part of $\mathbb{C}$. Let $\mathcal{A}$ be a sharp arc system over $\mathbb{C}$. For $\mathcal{X} \subseteq \mathcal{A}$, let $\mathcal{A}^{(\mathcal{X})}$ be the arc system that results from flipping all arcs in $\mathcal{X}$, i.e., $\mathcal{A}^{(\mathcal{X})} = (\mathcal{A} \setminus \mathcal{X}) \cup \{B^{\mathsf{F}} \mid B \in \mathcal{X}\}$. It is easy to see that $\mathcal{A}^{(\mathcal{X})}$ is an interval system if $\mathcal{X} = \{B \in \mathcal{A} \mid x \in B\}$ for some point $x \in \mathbb{C}$: If B_x is the arc that has x as extreme point, no arc in $\mathcal{A}^{(\mathcal{X})}$ contains both x and the point next to x that is contained in B_x.

There is a subtle difference between flipping and complementing arcs that contain a certain point; recall that the latter was used in Section 5.1 to reduce canonical representation of CA hypergraphs to that of interval hypergraphs. This difference arises when considering discrete circles, as is done in this thesis: Consider for example two disjoint arcs A and B whose union is the whole circle. Then their complements $\overline{A} = B$ and $\overline{B} = A$ are again disjoint from each other, while flipping them results in $A^{\mathsf{F}} \subseteq B^{\mathsf{F}}$.

McConnell observed [McC03] that flipping arcs of a sharp arc system $\mathcal{A}$ corresponds to the replacements in the intersection matrix $\mu_{\mathcal{A}}$ (cf. Definition 5.2.1) that are illustrated in Figure 5.3. These replacement rules can be applied to any CA matrix $\mu = (\mu_{i,j})_{i \neq j \in V}$. Let $\mu^{(X)}$ denote the result of flipping a subset X of the vertices of μ. Given a sharp arc representation $\alpha \colon V \to \mathcal{A}$ of μ, define the flipped representation $\alpha^{(X)} \colon V \to \mathcal{A}^{(\alpha(X))}$ by

$$\alpha^{(X)}(v) = \begin{cases} \alpha(v) & \text{if } v \notin X \\ (\alpha(v))^{\mathsf{F}} & \text{if } v \in X\,. \end{cases} \tag{5.1}$$

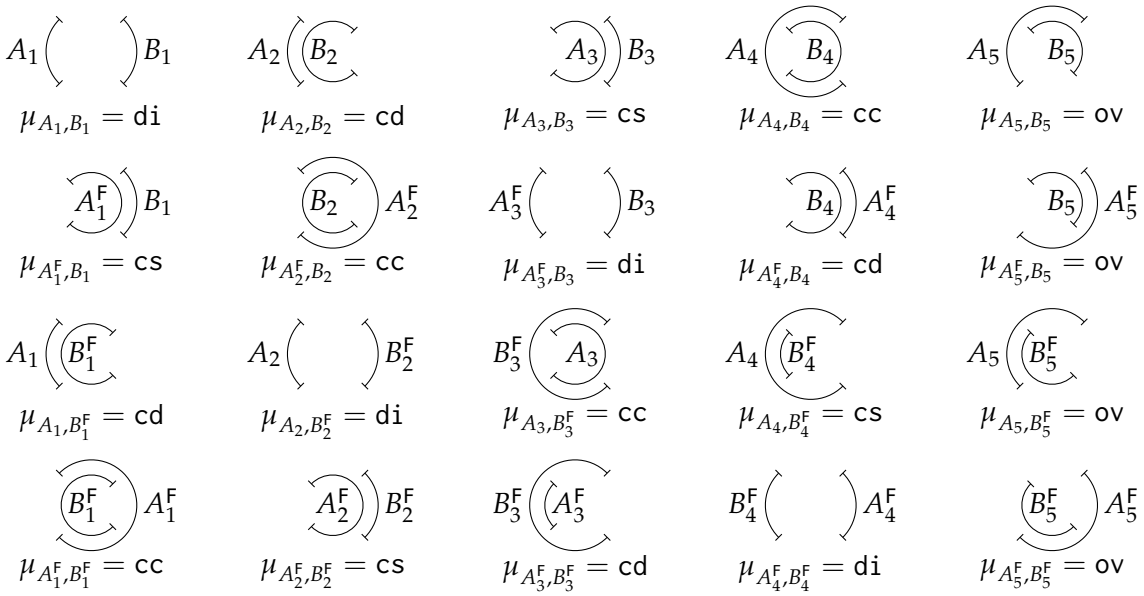

Figure 5.3: The effect of flipping arcs of a sharp arc system $\mathcal{A}$ on the entries of its intersection matrix $\mu_\mathcal{A} = (\mu_{A,B})_{A \neq B \in \mathcal{A}}$.

For a family $O = (O_u)_{u \in V}$, where O_u is an equivalence relation on $\{v \in V \mid \mu_{u,v} = \mathtt{ov}\}$ with two equivalence classes, define the flipped family $O^{(X)} = ({O_u}^{(X)})_{u \in V}$, which has $(v_1, v_2) \in {O_u}^{(X)}$ if and only if an odd number of the three statements $(v_1, v_2) \in O_u$, $v_1 \in X$, and $v_2 \in X$ holds. In other words, v_1 and v_2 remain (in)equivalent if and only if either none or both of them are flipped.

Lemma 5.2.8. *Let $\alpha \colon V \to \mathcal{A}$ be a sharp O-respecting arc representation of a CA matrix $\mu = (\mu_{u,v})_{u \neq v \in V}$ and let $X \subseteq V$. Then $\alpha^{(X)}$ is a sharp $O^{(X)}$-respecting arc representation of $\mu^{(X)}$.*

Proof. McConnell has shown that if $\alpha \colon V \to \mathcal{A}$ is a sharp arc representation of a CA matrix $\mu = (\mu_{u,v})_{u \neq v \in V}$ and $X \subseteq V$, then $\alpha^{(X)}$ is a sharp arc representation of $\mu^{(X)}$ [McC03]. It remains to show that $\alpha^{(X)}$ is $O^{(X)}$-respecting if α is O-respecting. Indeed, flipping $\alpha(u)$ does not change its extreme points and thus has no influence on O_u, while for $v \in V$ with $\mu_{u,v} = \mathtt{ov}$, the arcs $\alpha(v)$ and $\alpha(v)^\mathsf{F}$ contain the opposite extreme points of $\alpha(u)$. □

To ensure that the resulting matrix $\mu^{(X)}$ is interval, a suitable vertex set $X \subseteq V$ has to be used. As the algorithm does not have an arc representation of μ, it must identify the set X only from the structure of μ. If μ is HCA, the underlying graph G_μ is also HCA. The following fact implies that any maxclique C of G_μ can be used as X in this case.

Lemma 5.2.9. *Let $\alpha \colon \mathrm{V}(G) \to \mathcal{A}$ be any HCA representation of a graph G and let C be any maxclique of G. Then there is a point $p_C \in \mathrm{V}(\mathcal{A})$ with $\{\alpha(v) \mid v \in C\} = \{B \in \mathcal{A} \mid p_C \in B\}$.*

Proof. As C is a clique, the arcs in $\alpha(C) = \{\alpha(v) \mid v \in C\}$ intersect pairwise. As $\mathcal{A}$ is Helly, $\bigcap_{v \in C} \alpha(v)$ is non-empty. By maximality of C, no further arc can contain any point p_C in this intersection. □

In an interval graph, all maxcliques can be characterized as the common neighborhood of two vertices. This property was used in Section 4.1 to give a logspace reduction

from computing canonical representations of interval graphs to computing canonical representations of interval hypergraphs. This approach is not possible for HCA graphs, as they may contain maxcliques that cannot be characterized as the intersection or difference of constantly many neighborhoods; see Figure 5.4(a) for an example. However, at least one maxclique can be found in this way.

Theorem 5.2.10. *Let G be an HCA graph. Then there are $u, v \in V(G)$ (possibly $u = v$) such that $N[u, v]$ is a maxclique of G.*

Note that general CA graphs do not necessarily have such a maxclique, see Figure 5.4(b) for an example.

Proof. Let $\lambda_G = (\lambda_{u,v})_{u \neq v \in V(G)}$ be the neighborhood matrix of G and let $\alpha \colon V(G) \to \mathcal{A}$ be a normalized HCA representation of G; such an α exists by Lemmas 5.2.3 and 5.2.4. Choose v as a vertex such that there is no vertex w with $\lambda_{v,w} = \mathsf{cs}$ (i.e., $\nexists w : N[w] \subsetneq N[v]$). Note that there cannot be a vertex w' with $\lambda_{v,w'} = \mathsf{cc}$, since this would imply that there is a vertex $w \in N[v] \setminus N[w']$ (because $N[w'] \not\subset N[v]$) with $N[w] \subsetneq N[v]$ (equality can be ruled out because $w' \in N[v] \setminus N[u']$).

In case there is no vertex w with $\lambda_{v,w} = \mathsf{ov}$, $N[v]$ is a maxclique. This follows since $\lambda_{v,w} = \mathsf{cd}$ for all $w \in N(v)$ and hence, for all $w, w' \in N[v]$ it holds that $w \in N[v] \subseteq N[w']$.

Otherwise, choose a vertex $u \in N[v]$ with $\lambda_{v,u} = \mathsf{ov}$ such that $N[u, v]$ is minimal w.r.t. inclusion. To show that $N[u, v]$ is a maxclique, assume to the contrary that there exist $w, w' \in N[u, v]$ such that $w \notin N[w']$. If $\lambda_{v,w} = \mathsf{cd}$ (or $\lambda_{v,w'} = \mathsf{cd}$) then it follows that $w' \in N[v] \subseteq N[w]$ (or $w \in N[v] \subseteq N[w']$), a contradiction.

If $\lambda_{v,w} = \lambda_{v,w'} = \mathsf{ov}$ then $\alpha(w) \cap \alpha(w') = \emptyset$ and $\alpha(w) \not\subset \alpha(v) \not\subset \alpha(w')$. Since $\alpha(u) \not\subset \alpha(v)$ it follows that $\alpha(u)$ strictly overlaps $\alpha(v)$ from the same side as one of $\alpha(w)$ and $\alpha(w')$, say $\alpha(w)$; see Figure 5.5. It suffices to show that $N[w, v] \subsetneq N[u, v]$, because this yields a contradiction to the choice of u. As $w' \in N[u, v] \setminus N[w, v]$, it remains to show that

Figure 5.4: (a) Let G_n denote the split graph on $n + n$ vertices consisting of an n-clique C and a set S of n independent vertices, which are connected by the bipartite complement of a perfect matching between C and S. Every G_n is HCA; the figure shows an HCA model of G_4. Note that G_n has exactly $n + 1$ maxcliques, each of size n, and the maxclique C cannot be described as intersection or difference of less than n neighborhoods.
(b) The complement graph H_n of n independent edges is CA. It has 2^n maxcliques C_i, each containing exactly one vertex of each edge in $\overline{H_n}$. Since the common neighborhood of fewer than n vertices of H_n contains both vertices of at least one edge in $\overline{H_n}$, no maxclique C_i can be described in this way. The figure shows a CA model of H_4.

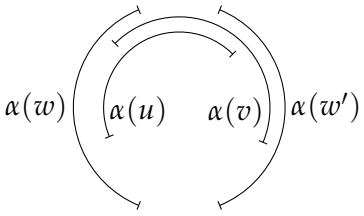

Figure 5.5: Proof of Theorem 5.2.10.

any vertex $x \in \mathrm{N}[w, v]$ is also contained in $\mathrm{N}[u, v]$. Indeed, the arcs $\alpha(w)$, $\alpha(v)$ and $\alpha(x)$ intersect pairwise, so the Helly property implies that there is a point p_x in $\alpha(w) \cap \alpha(v) \cap \alpha(x)$. Similarly, the Helly property implies $\alpha(u) \cap \alpha(v) \cap \alpha(w') \neq \emptyset$. As $\alpha(u)$ and $\alpha(w)$ strictly overlap $\alpha(v)$ from the same side, it follows that $\alpha(w) \cap \alpha(v) \subsetneq \alpha(u) \cap \alpha(v)$. Thus $p_x \in \alpha(u) \cap \alpha(v)$, which implies $x \in \mathrm{N}[u, v]$ as desired. □

5.2.3 Canonical representation of HCA matrices

Given an HCA matrix $\mu = (\mu_{u,v})_{u \neq v \in V}$ that admits an O-respecting arc representation, Theorem 5.2.10 and Lemma 5.2.9 allow to find $X \subseteq V$ so that the matrix $\mu^{(X)}$ admits an $O^{(X)}$-respecting interval representation. Theorem 4.6.3 allows to find such an interval representation $\alpha \colon V \to \mathcal{I}$ of $\mu^{(X)}$ in logspace, and shows that any sharp $O^{(X)}$-respecting interval model $\mathcal{I}'$ of $\mu^{(X)}$ is isomorphic to $\mathcal{I}$. Let $\mathcal{A}$ and $\mathcal{A}'$ be the sharp arc models of μ that can be obtained from $\mathcal{I}$ and $\mathcal{I}'$ by flipping the intervals of vertices in X. The following lemma will be used to show that any isomorphism from $\mathcal{I}$ to $\mathcal{I}'$ is also an isomorphism from $\mathcal{A}$ to $\mathcal{A}'$.

Lemma 5.2.11. *Let $\mathcal{I}$ and $\mathcal{J}$ be isomorphic sharp interval systems. For every hypergraph isomorphism φ from $\mathcal{I}$ to $\mathcal{J}$ there is a hypergraph isomorphism φ' from $\mathcal{I}$ to $\mathcal{J}$ such that $\varphi'(A) = \varphi(A)$ for all $A \in \mathcal{I}$ and, moreover, φ' respects extreme points, that is, takes the extreme points of each interval $A \in \mathcal{I}$ to the extreme points of the interval $\varphi(A) \in \mathcal{J}$.*

Proof. The argument is by induction on the number of overlap components of $\mathcal{I}$. In the base case, $\mathcal{I}$ and $\mathcal{J}$ are overlap-connected. Let $I_1, \ldots, I_k$ be the slots of $\mathcal{I}$ in the order they appear on the line, and let $J_1, \ldots, J_k$ be the slots of $\mathcal{J}$, also listed in ascending order. By Lemma 4.2.1, the isomorphism φ from $\mathcal{I}$ to $\mathcal{J}$ either maps each I_s onto J_s or it maps each I_s onto J_{k+1-s}. Assume the former; the latter case is symmetric.

We will see that, for each interval $A \in \mathcal{I}$, the isomorphism φ either respects the extreme points of A or can be locally modified to respect them. For each interval $A = [a^-, a^+]$, there are indices p and q such that $A = \bigcup_{s=p}^{q} I_s$. It follows that $\varphi(A) = \bigcup_{s=p}^{q} J_s$. Moreover, as $a^- \in I_p$ and $a^+ \in I_q$, it follows from $\varphi(A) = [b^-, b^+]$ that $b^- \in J_p$ and $b^+ \in J_q$.

Notice now that, since $\mathcal{I}$ is sharp, every slot contains at most two points. Moreover, every two-point slot $[c^-, d^+]$ consists of the start point of some interval C and the end point of another interval D. The transposition of the points c^- and d^+ violates neither C nor D, nor any other interval.

If I_p is a one-point slot, it immediately follows that $\varphi(a^-) = b^-$. So suppose that $I_p = [a^-, x^+]$ is a two-point slot. Let $J_p = [b^-, y^+]$. If $\varphi(a^-) = b^-$, there is nothing to do. Otherwise, it can be ensured that $\varphi'(a^-) = b^-$ by changing φ only on I_p.

A similar argument allows to ensure that $\varphi'(a^+) = b^+$, possibly modifying φ on I_q. In fact, it suffices to inspect each two-point slot once; if φ needs modification on this slot, this will simultaneously ensure that both points will be mapped to an extreme point of the correct interval. This completes the analysis of the overlap-connected case.

Suppose now that $\mathcal{I}$ and $\mathcal{J}$ have more than one overlap component, that is, are not overlap-connected. If the intersection graphs $\mathbb{I}(\mathcal{I})$ and $\mathbb{I}(\mathcal{J})$ are disconnected, the claim readily follows by applying the induction hypothesis to the corresponding connected components of $\mathcal{I}$ and $\mathcal{J}$.

It remains to consider the case when $\mathcal{I}$ and $\mathcal{J}$ are connected but not overlap-connected. Assume that an interval $A \in \mathcal{I}$ contains an inner overlap component $\mathcal{S} \subset \mathcal{I}$, then $\varphi(\operatorname{supp}(\mathcal{S})) \subset \varphi(A)$ for any isomorphism φ from $\mathcal{I}$ to $\mathcal{J}$. After removing all points in $\operatorname{supp}(\mathcal{S})$ from $\mathcal{I}$ and all points in $\varphi(\operatorname{supp}(\mathcal{S}))$ from $\mathcal{J}$, the resulting interval systems $\mathcal{I}'$ and $\mathcal{J}'$ will still contain the extreme points of A and $\varphi(A)$ respectively, and φ will induce an isomorphism from $\mathcal{I}'$ to $\mathcal{J}'$. By the induction hypothesis, there are isomorphisms from $\mathcal{I}'$ to $\mathcal{J}'$ and from $\mathcal{S}$ to $\varphi(\mathcal{S})$ that agree with φ on hyperedges and respect extreme points. Merging them, we get the desired isomorphism φ' from $\mathcal{I}$ to $\mathcal{J}$. □

Theorem 5.2.12. *Given an HCA matrix $\mu = (\mu_{u,v})_{u \neq v \in V}$ and a family $O = (O_u)_{u \in V}$ of equivalence relations, a sharp O-respecting arc representation $\alpha\colon V \to \mathcal{A}$ of μ can be computed in logspace; if that is not possible, the algorithm detects this. Moreover, for any sharp O-respecting arc representation $\alpha'\colon V \to \mathcal{A}'$ of μ, there is a hypergraph isomorphism φ from $\mathcal{A}$ to $\mathcal{A}'$ with $\varphi\big(\alpha(v)\big) = \alpha'(v)$ for all $v \in V$.*

Proof. The algorithm first computes the underlying graph G_μ and finds $u, v \in V$ such that $\mathrm{N}[u, v]$ is a maxclique; this search always succeeds by Theorem 5.2.10. It then flips the vertices in $X = \mathrm{N}[u, v]$, obtaining the family $O^{(X)}$ and the matrix $\mu^{(X)}$, which is interval by Lemma 5.2.9. Next, it invokes the logspace algorithm of Theorem 4.6.3 to compute a sharp $O^{(X)}$-respecting interval representation $\alpha_0\colon V \to \mathcal{I}$ of $\mu^{(X)}$ – or detects that none exists, in which case it rejects the input, which is correct by Lemma 5.2.8. Finally, the algorithm computes $\alpha = {\alpha_0}^{(X)}$ (cf. equation (5.1)), which is a sharp O-respecting arc representation of $\mu = (\mu^{(X)})^{(X)}$ by Lemma 5.2.8.

To prove the *moreover* part, let $\alpha'\colon V \to \mathcal{A}'$ be any sharp O-respecting arc representation of μ. Flipping the arcs $\mathcal{X} = \big\{\alpha'(v) \,\big|\, v \in X\big\}$ corresponding to the maxclique X results in the sharp $O^{(X)}$-respecting interval representation ${\alpha'}^{(X)}\colon V \to \mathcal{A}'^{(\mathcal{X})}$ of $\mu^{(X)}$. By Theorem 4.6.3, there is a hypergraph isomorphism φ from $\mathcal{I}$ to $\mathcal{I}' = \mathcal{A}'^{(\mathcal{X})}$ with $\varphi\big(\alpha_0(v)\big) = {\alpha'}^{(X)}(v)$ for all $v \in V$. By Lemma 5.2.11, it can be assumed that φ respects extreme points. This implies that, for any interval $B = [b^-, b^+]$ in $\mathcal{I}$, this isomorphism φ maps $B^{\mathsf{F}} = \overline{B} \cup \big\{b^-, b^+\big\}$ to $\big(\varphi(B)\big)^{\mathsf{F}} = \overline{\varphi(B)} \cup \big\{\varphi(b^-), \varphi(b^+)\big\}$. Thus φ is also an isomorphism from $\mathcal{A}$ to $\mathcal{A}'$ with $\varphi\big(\alpha(v)\big) = \alpha'(v)$ for all $v \in V$. □

Combining this algorithm with the logspace algorithm for canonical representation of CA hypergraphs of Theorem 5.1.1 allows canonical O-respecting representation of HCA matrices in logspace. To ensure that the resulting representations are sharp, the following lemma is needed.

Lemma 5.2.13. *Canonical sharp arc representations of edge-colored sharp arc systems can be computed in logspace. The computed representations preserve extreme points.*

Proof. Given a sharp arc system $\mathcal{A}$ with an edge coloring $c_{\mathcal{A}} \colon \mathcal{A} \to \mathbb{N}^+$, the algorithm first constructs the hypergraph $\mathcal{H}$ defined by

$$\begin{aligned} \mathrm{V}(\mathcal{H}) &= \{p, p' \mid p \in \mathrm{V}(\mathcal{A})\} \text{ and} \\ \mathcal{H} &= \{B \cup \{p' \mid p \in B \setminus \{b^-, b^+\}\} \mid B = [b^-, b^+] \in \mathcal{A}\} \\ &\quad \cup \{\{p, p'\} \mid p \in \mathrm{V}(\mathcal{A})\}, \end{aligned}$$

where each hyperedge $B' \in \mathcal{H}$ with $B' \cap \mathrm{V}(\mathcal{A}) \in \mathcal{A}$ has multiplicity $c_{\mathcal{A}}(B' \cap \mathrm{V}(\mathcal{A}))$ to encode the colors.

Note that $\mathcal{H}$ is a CA hypergraph, as each new point p' can be inserted next to p on the side that is not contained in the arc that has p as extreme point.

Next, the algorithm of Theorem 5.1.1 is invoked to compute a canonical arc representation $\rho_{\mathcal{H}} \colon \mathrm{V}(\mathcal{H}) \to [1, |\mathrm{V}(\mathcal{H})|]$ of $\mathcal{H}$. Finally, the restriction $\rho_{\mathcal{A}}$ of $\rho_{\mathcal{H}}$ to $\mathrm{V}(\mathcal{A})$ that preserves the circular order of the points $\{\rho_{\mathcal{H}}(p) \mid p \in \mathrm{V}(\mathcal{A})\}$ is returned.

Clearly, $\rho_{\mathcal{A}}$ maps each arc $B \in \mathcal{A}$ to circularly consecutive points. Also, it has to map the extreme points of B to the extreme points of $\rho_{\mathcal{A}}(B)$ because of the hyperedges $\{p, p'\}$, $p \in \mathrm{V}(\mathcal{A})$. It remains to show that isomorphic arc systems receive equal arc models. If there is a color-preserving isomorphism from $\mathcal{A}$ to $\mathcal{A}'$, the hypergraphs $\mathcal{H}$ and $\mathcal{H}'$ are isomorphic. It then follows from Theorem 5.1.1 that $\rho_{\mathcal{H}}(\mathcal{H}) = \rho_{\mathcal{H}'}(\mathcal{H}')$. Now observe that a vertex $p \in \mathrm{V}(\mathcal{H})$ belongs to $\mathrm{V}(\mathcal{A})$ if and only if there is a hyperedge $\{p, p'\} \in \mathcal{H}$ such that all but one hyperedges that contain p also contain p', and p' is contained in no other hyperedge. Thus the same points are omitted from both arc models, showing $\rho_{\mathcal{A}}(\mathcal{A}) = \rho_{\mathcal{A}'}(\mathcal{A}')$. The multiplicities of the hyperedges of $\mathcal{H}$ ensure that $c_{\mathcal{A}}(\rho_{\mathcal{A}}^{-1}(B)) = c_{\mathcal{A}'}(\rho_{\mathcal{A}'}^{-1}(B))$ for all arcs $B \in \rho_{\mathcal{A}}(\mathcal{A})$. □

Lemma 5.2.14. *Canonical sharp O-respecting arc representations for vertex-colored HCA matrices can be computed in logspace.*

Proof. Let $\mu = (\mu_{u,v})_{u \neq v \in V}$ be an HCA matrix, let $O = (O_u)_{u \in V}$ be a family of equivalence relations, and let $c \colon V \to C$ be a coloring of its vertices. The first step is to invoke the algorithm of Theorem 5.2.12 to obtain a sharp O-respecting arc representation $\alpha_\mu \colon V \to \mathcal{A}_\mu$ of μ, or to detect that none exists. As a second step, the algorithm of Lemma 5.2.13 is used to compute a canonical sharp arc representation $\rho_\mu \colon \mathrm{V}(\mathcal{A}_\mu) \to [1, 2 \cdot |V|]$ of $\mathcal{A}_\mu$, where each arc $\alpha_\mu(v) \in \mathcal{A}_\mu$ is colored with $c(v)$. Finally, the algorithm returns the arc representation of μ defined by $v \mapsto \rho_\mu(\alpha_\mu(v))$. Note that this arc representation is O-respecting, as ρ_μ preserves extreme points.

It remains to show canonicity. Let $\mu' = (\mu'_{u,v})_{u \neq v \in V'}$ be a CA matrix isomorphic to μ, and let $\psi \colon V \to V'$ be an isomorphism from μ to μ'. Then α_μ and $\alpha_{\mu'} \circ \psi$ are both sharp arc representations of μ. By Theorem 5.2.12, there is an isomorphism φ between the resulting arc models $\mathcal{A}_\mu$ and $\mathcal{A}_{\mu'}$ with $\varphi(\alpha_\mu(v)) = \alpha_{\mu'}(\psi(v))$ for all $v \in V$. This implies that φ preserves colors. So μ and μ' receive equal colored arc models $\rho_\mu(\mathcal{A}_\mu) = \rho_{\mu'}(\mathcal{A}_{\mu'})$. □

Together with Lemma 5.2.7, this gives the main result for HCA graphs.

Theorem 5.2.15. *There is a logspace algorithm that computes canonical HCA representations for HCA graphs.*

As interval graphs are a subclass of HCA graphs, this implies together with the hardness result from Section 4.7 that isomorphism of HCA graphs is L-complete.

5.3 Canonical representation of proper circular-arc graphs and concave-round graphs

In this section, a logspace algorithm is presented that computes canonical arc representations of concave-round graphs. When the input graphs are proper circular-arc, the resulting arc models are proper.

Bang-Jensen, Huang, and Yeo [BHY00] call a graph G *concave-round* (resp. *convex-round*) if $\mathcal{N}[G]$ (resp. $\mathcal{N}(G)$) is a CA hypergraph. Since $\overline{\mathcal{N}[G]} = \mathcal{N}(\overline{G})$, concave-round and convex-round graphs are co-classes. Using this terminology, a result of Tucker [Tuc71] says that PCA graphs are concave-round, and concave-round graphs are CA.

Let G be a graph and let $\alpha\colon \mathrm{V}(G) \to \mathcal{A}$ be an arc representation of G. If $\mathrm{V}(\mathcal{A}) \notin \mathcal{A}$ (this always holds when G has no universal vertex), the lifted circular order $\prec_{\mathcal{A}}$ on $\mathcal{A}$ (see Section 2.4) can be used to define a circular order $\prec_\alpha$ on $\mathrm{V}(G)$, where $u \prec_\alpha v$ if and only if $\alpha(v) \prec_{\mathcal{A}} \alpha(u)$. The circular order $\prec_\alpha$ is called the geometric order on $\mathrm{V}(G)$ associated with α.

The connections of PCA and concave-round graphs to CA hypergraphs are outlined in Section 5.3.1. In particular, it will be useful that the neighborhood hypergraph $\mathcal{N}[G]$ of a non-co-bipartite PCA graph G admits a unique CA order, which coincides with the geometric order $\prec_\alpha$ for any proper arc representation α of G. Based on this, Sections 5.3.2 and 5.3.3 explain how to compute canonical arc representations of non-co-bipartite PCA graphs in logspace. To achieve the same for co-bipartite PCA graphs G (and all non-PCA concave-round graphs), the fact that $\mathcal{N}(\overline{G})$ is in this case an interval hypergraph is used. Moreover, an interval representation of $\mathcal{N}(\overline{G})$ can be transformed into an arc representation of G; see Section 5.3.4 for details.

5.3.1 Linking PCA graphs and tight CA hypergraphs

To connect the canonical representation problem for PCA and concave-round graphs to that of CA hypergraphs, the graph classes under consideration are characterized in terms of neighborhood hypergraphs. For concave-round graphs, this directly follows from their definition, and accompanying hypergraphs can be found also for PCA graphs.

Theorem 5.3.1. *A graph G is PCA if and only if $\mathcal{N}[G]$ is a tight CA hypergraph.*

This characterization gives a logspace algorithm for recognition of PCA graphs: Given a graph G compute its neighborhood hypergraph $\mathcal{N}[G]$, its tightened version $(\mathcal{N}[G])^{\Subset}$ (cf. Section 2.4), and check whether the latter is CA using the algorithm of Theorem 5.1.1.

Corollary 5.3.2. *It can be checked in logspace whether a given graph is PCA.*

The forward direction of Theorem 5.3.1 follows from Lemma 5.3.3 below. To prove the other direction, distinguish two cases. If $\overline{G}$ is not bipartite, then a result of Tucker says that G is a PCA graph whenever $\mathcal{N}[G]$ is a CA hypergraph [Tuc71]. The case of bipartite $\overline{G}$ is treated in Section 5.3.4 where it is shown that any tight arc model for $\mathcal{N}[G]$ can in this case be transformed into a proper arc model for G. Thus, the proof of Theorem 5.3.1 will be completed in Section 5.3.4.

Lemma 5.3.3. *The geometric order $\prec_\alpha$ on $\mathrm{V}(G)$ associated with a proper arc representation α of a graph G is a tight CA order for the hypergraph $\mathcal{N}[G]$.*

Proof. Let G be a PCA graph and let $\alpha\colon V(G) \to \mathcal{A}$ be a proper arc representation of G. The first step is to show that, for each vertex $u \in V(G)$, the neighborhood $N[u]$ is an arc w.r.t. the order $\prec_\alpha$. If u is universal, the claim is trivial. Otherwise, let $\alpha(u) = [a^-, a^+]$ and split $N(u)$ in two parts, namely $N^-(u) = \{v \in N(u) \mid a^- \in \alpha(v)\}$ and $N^+(u) = \{v \in N(u) \mid a^+ \in \alpha(v)\}$. Indeed, no vertex v is contained in both $N^-(u)$ and $N^+(u)$. Otherwise, since $\mathcal{A}$ is proper, the arcs $\alpha(v)$ and $\alpha(u)$ would cover the whole circle, both intersecting any other arc $\alpha(w)$, contradicting the assumption that u is nonuniversal.

Now let $v \in N^+(u)$ and assume that $u \prec_\alpha v_1 \prec_\alpha \ldots \prec_\alpha v_k \prec_\alpha v$. To show that every vertex v_i is in $N^+(u)$, observe that the definition of $\prec_\alpha$ implies $\alpha(u) \prec_\mathcal{A} \alpha(v_1) \prec_\mathcal{A} \ldots \prec_\mathcal{A} \alpha(v_k) \prec_\mathcal{A} \alpha(v)$. If $\alpha(v) = [c^-, c^+]$ and $\alpha(v_i) = [b^-, b^+]$, we see that $b^- \in (a^-, c^-)$, $b^+ \in (a^+, c^+)$ and, hence, $a^+ \in [b^-, b^+]$. It follows that $N^+(u) \cup \{u\}$ is an arc starting at u. By a symmetric argument, $N^-(u) \cup \{u\}$ is an arc ending at u. Thus $N[u]$ is also an arc, implying that $\prec_\alpha$ is a CA order for $\mathcal{N}[G]$.

It remains to show that the CA order $\prec_\alpha$ of $\mathcal{N}[G]$ is tight. Suppose that $N[u] = [u^-, u^+] \subseteq N[v] = [v^-, v^+]$ and v is nonuniversal with $\alpha(v) = [c^-, c^+]$. First assume that $u \in N^+(v) = (v, v^+]$. Since $u, v^+ \in N^+(v)$, it follows that $c^+ \in \alpha(u) \cap \alpha(v^+)$. Hence, u and v^+ are adjacent or equal, which implies that $u^+ = v^+$. If $u \in [v^-, v)$, a symmetric argument shows that $u^- = v^-$. □

Theorem 5.3.1 suggests that a tight CA order of $\mathcal{N}[G]$ can be used to construct a proper arc model for G. For this, the converse of Lemma 5.3.3 is needed. In the case that $\overline{G}$ is not bipartite, the following proposition implies that indeed each CA order of $\mathcal{N}[G]$ is the geometric order of some proper arc representation of G.

Proposition 5.3.4 [Hua95, Theorem 4.5]**.** *If G is a connected twin-free PCA graph and $\overline{G}$ is not bipartite, then $\mathcal{N}[G]$ has a unique CA order up to reversing.*

Closing this section, co-bipartite concave-round graphs G are characterized using properties of $\mathcal{N}(\overline{G})$. Given a bipartite graph H and a bipartition $V(H) = U \cup W$ of its vertices into two independent sets, let $\mathcal{N}_U(H)$ denote the hypergraph $\{\!\{N(w) \mid w \in W\}\!\}$ on the vertex set U. Note that $\mathcal{N}_U(H)$ and $\mathcal{N}_W(H)$ are dual hypergraphs, i.e., $(\mathcal{N}_U(H))^D \cong \mathcal{N}_W(H)$. Recall that a bipartite graph H is called convex if its vertex set admits splitting into two independent sets U and W, such that $\mathcal{N}_U(H)$ is an interval hypergraph. Similarly, if both $\mathcal{N}_U(H)$ and $\mathcal{N}_W(H)$ are interval hypergraphs, H is called biconvex [Spi03]. As G is co-bipartite concave-round if and only if its complement $H = \overline{G}$ is bipartite convex-round, the following fact gives the desired characterization.

Proposition 5.3.5 [Tuc74, Theorem 2.2]**.** *A graph H is bipartite convex-round if and only if it is biconvex and if and only if $\mathcal{N}(H)$ is an interval hypergraph.*

5.3.2 A strategy for canonical representation

In this and the following two sections, the canonical representation algorithm for concave-round and PCA graphs is described. For a given graph, it has to compute an arc representation such that the resulting arc models are equal for isomorphic input graphs.

Theorem 5.3.6. *There is a logspace algorithm that computes canonical arc representations for the class of concave-round graphs. Moreover, this algorithm outputs a proper arc representation whenever the input graph is PCA.*

As proper interval graphs are a subclass of PCA graphs and thus of concave-round graphs, Theorem 4.7.5 implies that isomorphism of these classes is L-complete.

For any class of intersection graphs, a canonical representation algorithm readily implies a canonical labeling algorithm of the same complexity. Vice versa, a canonical representation algorithm follows from a canonical labeling algorithm *and* a representation algorithm (not necessarily a canonical one). Proving Theorem 5.3.6 according to this scheme, splits the task in two parts: First compute a canonical labeling λ of the input graph G and then compute an arc representation α of the canonical form $\lambda(G)$. Then the composition $\alpha \circ \lambda$ is a canonical arc representation of G. As twins can be easily re-inserted in a (proper) arc representation, it suffices to compute α for the quotient graph of $\lambda(G)$, which has only one vertex for each twin class.

The algorithm distinguishes two cases depending on whether $\overline{G}$ is bipartite; see Figure 5.6 for an overview of the involved graph classes.

5.3.3 Non-co-bipartite concave-round graphs

As mentioned before, any concave-round graph G whose complement is not bipartite is actually a PCA graph [Tuc71]. Hence, the computed arc representation must be proper in this case.

Canonical labeling
First transform G into its quotient graph G', which has one vertex for each twin class $[v]$ of G. Let n be the number of vertices in G'. Then use the algorithm given by Theorem 5.1.1 to compute an arc representation ρ of $\mathcal{N}[G']$. By Proposition 5.3.4, $\mathcal{N}[G']$ has a CA order which is unique up to reversing. Hence, $\mathcal{N}[G']$ admits at most $2n$ different arc representations $\rho_1, \ldots, \rho_{2n}$, which can be obtained from ρ by cyclic shifts and reversing. Each of these specifies a labeling $\lambda_i \colon V(G) \to [1, |V(G)|]$ of G with $\lambda_i(u) < \lambda_i(v) \Leftrightarrow \rho_i([u]) < \rho_i([v])$ up to permutation of twins. As permutations of twins do not change the images $\lambda_i(G)$, one of these $2n$ variants that gives the lexicographically least canonical form $\lambda_i(G)$ of G can be appointed as the canonical labeling λ_G of G.

Proper arc representation
As mentioned above, it may be assumed that the given graph G is twin-free. Compute a CA order $\prec$ of $\mathcal{N}[G]$ using the algorithm of Theorem 5.1.1. By Lemma 5.3.3 and Proposition 5.3.4, there is a proper arc representation $\alpha \colon V(G) \to \mathcal{A}$ of G whose associated geometric order $\prec_\alpha$ coincides with $\prec$. In order to construct α from $\prec$, we can assume that

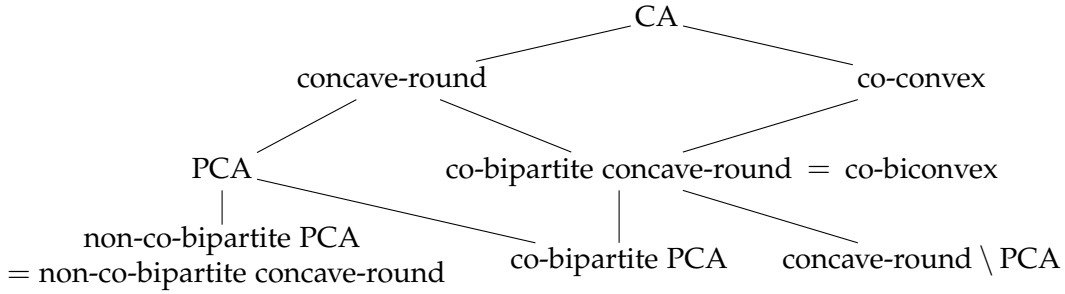

Figure 5.6: Hasse diagram of the inclusions between the considered CA graph classes.

$\mathcal{A}$ is sharp, i.e., that every point $p \in V(\mathcal{A})$ is the extreme point of exactly one arc in $\mathcal{A}$. A suitable circular order on $V(\mathcal{A})$ is uniquely determined by the conditions that the start points a_v^- appear on the circle according to $\prec$, the same holds true for the end points a_v^+, and that each end point a_v^+ lies between the start point $a_{v^+}^-$ and the following start point, where v^+ is the end point of the arc $N[v]$ w.r.t. $\prec$. Using this characterization, α can easily be computed in logspace. Note that the extreme points of $N[v] = [v^-, v^+]$ are well defined because no vertex v can be universal; otherwise the arcs containing the extreme points of $\alpha(v)$ would correspond to two cliques covering the whole vertex set $V(G)$.

5.3.4 Co-bipartite concave-round graphs

By Proposition 5.3.5, co-bipartite concave-round graphs are precisely the co-biconvex graphs. In fact, even all co-convex graphs are CA (this is implicit in Tucker's results [Tuc71]) and it is possible to compute canonical arc representations actually for this larger class of graphs.

Canonical labeling
The canonical labeling problem of co-convex graphs reduces to that of convex graphs, which can be solved in logspace by Corollary 4.3.2.

(Proper) arc representation
Let us first recall Tucker's argument [Tuc71] showing that, if the complement of G is a convex graph, then G is CA. We can assume that $\overline{G}$ has no fraternal vertices as those would correspond to twins in G.

Let $V(G) = U \cup W$ be a partition of $\overline{G}$ into independent sets such that $\mathcal{N}_U(\overline{G})$ is an interval hypergraph. Let $u_1, \ldots, u_k$ be an *interval order* on U for $\mathcal{N}_U(\overline{G})$, i.e., a linear order on U such that every hyperedge in $\mathcal{N}_U(\overline{G})$ is an interval w.r.t. this order. The algorithm constructs an arc representation α for G on the circle $\mathbb{C}_{2k+1}$ (see Figure 5.7 for an example) by setting $\alpha(u_i) = [i+1, i+k]$ for each $u_i \in U$ and $\alpha(w) = [j+k+1, i]$ for each $w \in W$, where $N_{\overline{G}}(w) = [u_i, u_j]$ and the subscript $\overline{G}$ means that the neighborhood is considered in the complement of G. Note that $\alpha(w) = \mathbb{C}_{2k+1} \setminus \bigcup_{u \in N_{\overline{G}}(w)} \alpha(u)$. In the

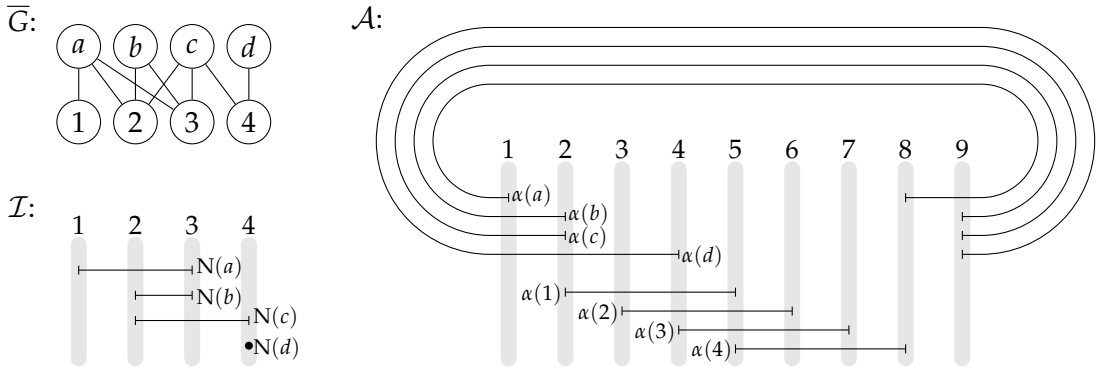

Figure 5.7: $\overline{G}$ is the complement of a co-bipartite concave-round graph G with the bipartition $U = \{1,2,3,4\}$ and $W = \{a,b,c,d\}$. The interval order corresponding to the interval model $\mathcal{I}$ of $\mathcal{N}_U(\overline{G})$ is used to construct the arc representation $\alpha\colon V(G) \to \mathcal{A}$ of G; see the text for details.

case that $N_{\overline{G}}(w) = \emptyset$, the algorithm sets $\alpha(w) = [1, k+1]$. By construction, all arcs $\alpha(u)$ for $u \in U$ share the point $k+1$, all arcs $\alpha(w)$ for $w \in W$ share the point 1, and any pair $\alpha(u)$ and $\alpha(w)$ is intersecting if and only if u and w are adjacent in G. Thus, α is indeed an arc representation for G.

In order to compute α in logspace, it suffices to compute a suitable bipartition $\{U, W\}$ of $\overline{G}$ and an interval order of the hypergraph $\mathcal{N}_U(\overline{G})$ in logspace. Finding a bipartition $\{U, W\}$ such that $\mathcal{N}_U(\overline{G})$ is an interval hypergraph can be done by splitting $\overline{G}$ into connected components $H_1, \ldots, H_k$ (using Reingold's algorithm [Rei08]) and finding such a bipartition $\{U_i, W_i\}$ for each component H_i. By Theorem 4.2.6, the algorithm can actually compute interval orders of the hypergraphs $\mathcal{N}_{U_i}(H_i)$ which can be easily pasted together to give an interval order of $\mathcal{N}_U(\overline{G})$. Together with the canonical labeling algorithm this implies that canonical arc representations for co-convex graphs and, in particular, for co-bipartite concave-round graphs can be computed in logspace.

It remains to show that for co-bipartite PCA graphs, there is a logspace algorithm that actually computes proper arc representations. The existence of such an arc representation will also complete the proof of Theorem 5.3.1. As above, we may assume that G is twin-free. By Lemma 5.3.3, the hypergraph $\mathcal{N}[G]$ has a tight CA order $\prec$. The algorithm computes $\prec$ in logspace by running the algorithm given by Theorem 5.1.1 on the tightened hypergraph $(\mathcal{N}[G])^{\Subset}$. Any tight CA order of $\mathcal{N}[G]$ is also a tight CA order of $\mathcal{N}(\overline{G}) = \overline{\mathcal{N}[G]}$. Let $V(G) = U \cup W$ be a bipartition of $\overline{G}$ into two independent sets. Note that the restriction of a tight CA order of $\mathcal{N}(\overline{G})$ to $\mathcal{N}_U(\overline{G})$ is a tight interval order of the interval hypergraph $\mathcal{N}_U(\overline{G})$. Retracing Tucker's construction of an arc representation α for a co-convex graph G (which is outlined above) in the case that the interval order of $\mathcal{N}_U(\overline{G})$ is tight, we see that α now gives us a tight arc model for G. Note that, by construction, this model contains no complete arc. It remains to observe that any tight α with this property can be converted into a proper arc representation α'. Tucker [Tuc71] described such a transformation, and Chen [Che97] observed that it can be implemented in AC^1. In this transformation, each point in the tight arc representation that is the extreme point of more than one arc is replaced by a sequence of points, each becoming the new extreme point of one of these arcs. The new points can be ordered so that none of these arcs is contained in another; see Figure 5.8 for an illustration. Similar ideas as in the proof of Lemma 4.6.1 allow to implement this construction also in logspace.

This completes the proof of Theorem 5.3.6 and additionally proves the following corollary.

Corollary 5.3.7. *There is a logspace algorithm that computes canonical arc representations for co-convex graphs.*

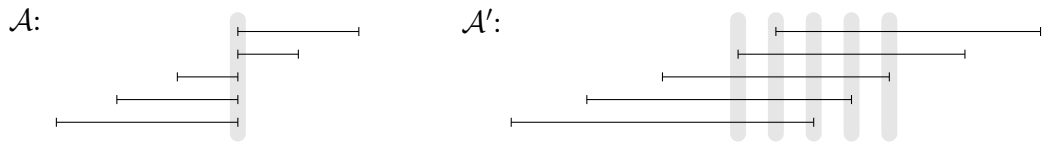

Figure 5.8: How to make a tight arc representation proper

Bibliography

[ADK+12] V. Arvind, Bireswar Das, Johannes Köbler, and Sebastian Kuhnert. 'The isomorphism problem for k-trees is complete for logspace'. In: *Information and Computation* 217 (Aug. 2012), pp. 1–11.

[ADK07] V. Arvind, Bireswar Das, and Johannes Köbler. 'The space complexity of k-tree isomorphism'. In: *Proceedings of 18th International Symposium on Algorithms and Computation (ISAAC)*. LNCS 4853. Berlin: Springer, 2007, pp. 822–833.

[AHU74] Alfred V. Aho, John Edward Hopcroft, and Jeffrey D. Ullman. *The design and analysis of computer algorithms*. Reading, Mass.: Addison Wesley, 1974. ISBN: 0-201-00029-6.

[ÁJ93] Carme Álvarez and Birgit Jenner. 'A very hard log-space counting class'. In: *Theoretical Computer Science* 107.1 (Jan. 1993), pp. 3–30.

[AK06] V. Arvind and Piyush P. Kurur. 'Graph isomorphism is in SPP'. In: *Information and Computation* 204.5 (May 2006), pp. 835–852.

[AM04] Eric Allender and Meena Mahajan. 'The complexity of planarity testing'. In: *Information and Computation* 139.1 (Feb. 2004).

[AMO93] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network flows. Theory, algorithms, and applications*. Englewood Cliffs, N.J.: Prentice Hall, 1993. ISBN: 0-13-617549-X.

[AS98] Fred Annexstein and Ram Swaminathan. 'On testing consecutive-ones property in parallel'. In: *Discrete Applied Mathematics* 88.1-3 (Nov. 1998), pp. 7–28.

[AT05] V. Arvind and Jacobo Torán. 'Isomorphism testing: Perspective and open problems'. In: *Bulletin of the European Association for Theoretical Computer Science* 86 (June 2005). Computational Complexity Column, pp. 66–84.

[Bab79] László Babai. *Monte-Carlo algorithms in graph isomorphism testing*. Tech. rep. 79–10. Université de Montréal, 1979.

[BC79] Kellogg S. Booth and Charles J. Colbourn. *Problems polynomially equivalent to Graph Isomorphism*. Tech. rep. CS-77-04. University of Waterloo, Computer Science Department, 1979.

[BDH+92] Gerhard Buntrock, Carsten Damm, Ulrich Hertrampf, and Christoph Meinel. 'Structure and importance of logspace-MOD class'. In: *Theory of Computing Systems* 25.3 (Sept. 1992), pp. 223–237.

[BGM82] László Babai, D. Yu. Grigoryev, and David M. Mount. 'Isomorphism of graphs with bounded eigenvalue multiplicity'. In: *Proceedings of 14th Annual ACM Symposium on Theory of Computing (STOC)*. 1982, pp. 310–324.

[BHI07] Jørgen Bang-Jensen, Jing Huang, and Louis Ibarra. 'Recognizing and representing proper interval graphs in parallel using merging and sorting'. In: *Discrete Applied Mathematics* 155.4 (Feb. 2007), pp. 442–456.

[BHY00] Jørgen Bang-Jensen, Jing Huang, and Anders Yeo. 'Convex-round and concave-round graphs'. In: *SIAM Journal on Discrete Mathematics* 13.2 (2000), pp. 179–193.

[BHZ87] Ravi B. Boppana, Johan Håstad, and Stathis Zachos. 'Does co-NP have short interactive proofs?' In: *Information Processing Letters* 25.2 (May 1987), pp. 127–132.

[BIS90] David A. Mix Barrington, Neil Immerman, and Howard Straubing. 'On uniformity within NC^1'. In: *Journal of Computer and System Sciences* 41.3 (Dec. 1990), pp. 274–306.

[BL75] Kellogg S. Booth and George S. Lueker. 'Linear algorithms to recognize interval graphs and test for the consecutive ones property'. In: *Proceedings of 7th Annual ACM Symposium on Theory of Computing (STOC)*. New York: ACM, 1975, pp. 255–265.

[BL76] Kellogg S. Booth and George S. Lueker. 'Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms'. In: *Journal of Computer and System Sciences* 13.3 (Dec. 1976), pp. 335–379.

[BL83] László Babai and Eugene M. Luks. 'Canonical labeling of graphs'. In: *Proceedings of 15th Annual ACM Symposium on Theory of Computing (STOC)*. 1983, pp. 171–183.

[BLS99] Andreas Brandstädt, Van Bang Le, and Jeremy P. Spinrad. *Graph classes. A survey*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 1999. ISBN: 0-89871-432-X.

[BO95] Luitpold Babel and Stephan Olariu. 'On the isomorphism of graphs with few P_4s'. In: *Proceedings of the 21st International Workshop on Graph-Theoretic Concepts in Computer Science*. LNCS 1017. Springer, 1995, pp. 24–36.

[Bod90] Hans L. Bodlaender. 'Polynomial algorithms for graph isomorphism and chromatic index on partial k-trees'. In: *Journal of Algorithms* 11.4 (Dec. 1990), pp. 631–643.

[Boo75] Kellogg S. Booth. 'PQ-tree algorithms'. PhD thesis. Department of Computer Science, University of California, Berkeley, 1975.

[BPT96] Luitpold Babel, Ilia N. Ponomarenko, and Gottfried Tinhofer. 'The isomorphism problem for directed path graphs and for rooted directed path graphs'. In: *Journal of Algorithms* 21.3 (Nov. 1996), pp. 542–564.

[BS65] Robert G. Busacker and Thomas L. Saaty. *Finite graphs and networks. An introduction with applications*. New York: McGraw-Hill, 1965.

[Bus97] Samuel R. Buss. 'Alogtime algorithms for tree isomorphism, comparison, and canonization'. In: *Computational Logic and Proof Theory. 5th Kurt Gödel Colloquium*. LNCS 1289. Berlin: Springer, 1997, pp. 18–33.

[Che93] Lin Chen. 'Efficient parallel recognition of some circular arc graphs, I'. In: *Algorithmica* 9.3 (1993), pp. 217–238.

[Che96] Lin Chen. 'Graph isomorphism and identification matrices: Parallel algorithms'. In: *IEEE Transactions on Parallel and Distributed Systems* 7.3 (Mar. 1996), pp. 308–319.

[Che97] Lin Chen. 'Efficient parallel recognition of some circular arc graphs, II'. In: *Algorithmica* 17.3 (1997), pp. 266–280.

[Che99] Lin Chen. 'Graph isomorphism and identification matrices: Sequential algorithms'. In: *Journal of Computer and System Sciences* 59.3 (Dec. 1999), pp. 450–475.

[CI88] N. Chandrasekharan and S. Sitharama Iyengar. 'NC algorithms for recognizing chordal graphs and k trees'. In: *IEEE Transactions on Computers* 37.10 (Oct. 1988), pp. 1178–1183.

[CKN+95] Derek G. Corneil, Hiryoung Kim, Sridhar Natarajan, Stephan Olariu, and Alan P Sprague. 'Simple linear time recognition of unit interval graphs'. In: *Information Processing Letters* 55.2 (July 1995), pp. 99–104.

[CLM+13] Andrew R. Curtis, Min Chih Lin, Ross M. McConnell, Yahav Nussbaum, Francisco J. Soulignac, Jeremy P. Spinrad, and Jayme L. Szwarcfiter. 'Isomorphism of graph classes related to the circular-ones property'. In: *Discrete Mathematics & Theoretical Computer Science* 15.1 (2013), pp. 157–182.

[Cor04] Derek G. Corneil. 'A simple 3-sweep LBFS algorithm for the recognition of unit interval graphs'. In: *Discrete Applied Mathematics* 138.3 (Apr. 2004), pp. 371–379.

[COS09] Derek G. Corneil, Stephan Olariu, and Lorna Stewart. 'The LBFS structure and recognition of interval graphs'. In: *SIAM Journal on Discrete Mathematics* 23.4 (2009), pp. 1905–1953.

[CY91] Lin Chen and Yaacov Yesha. 'Parallel recognition of the consecutive ones property with applications'. In: *Journal of Algorithms* 12.3 (Sept. 1991), pp. 375–392.

[CY93] Lin Chen and Yaacov Yesha. 'Efficient parallel algorithms for bipartite permutation graphs'. In: *Networks* 23.1 (1993), pp. 29–39.

[DDN13] Bireswar Das, Samir Datta, and Prajakta Nimbhorkar. 'Log-space algorithms for paths and matchings in k-trees'. In: *Theory of Computing Systems* 53.4 (Nov. 2013), pp. 669–689.

[DHH96] Xiaotie Deng, Pavol Hell, and Jing Huang. 'Linear-time representation algorithms for proper circular-arc graphs and proper interval graphs'. In: *SIAM Journal on Computing* 25.2 (Apr. 1996), pp. 390–403.

[DLN+09] Samir Datta, Nutan Limaye, Prajakta Nimbhorkar, Thomas Thierauf, and Fabian Wagner. 'Planar graph isomorphism is in log-space'. In: *Proceedings of 24th Annual IEEE Conference on Computational Complexity (CCC)*. IEEE Computer Society, 2009, pp. 203–214.

[DNT+09] Samir Datta, Prajakta Nimbhorkar, Thomas Thierauf, and Fabian Wagner. 'Graph isomorphism for $K_{3,3}$-free and K_5-free graphs is in log-space'. In: *Proceedings of 29th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*. LIPIcs 4. Dagstuhl: Leibniz-Zentrum für Informatik, 2009, pp. 145–156.

[DTW12] Bireswar Das, Jacobo Torán, and Fabian Wagner. 'Restricted space algorithms for isomorphism on bounded treewidth graphs'. In: *Information and Computation* 217 (Aug. 2012), pp. 71–83.

[Duc78] Pierre Duchet. « Propriété de Helly et problèmes de représentation ». In : *Problèmes Combinatoires et Théorie des Graphes (Orsay, 1976)*. Colloquium International CNRS 260. Paris : CNRS, 1978, p. 117–118.

[EJT10] Michael Elberfeld, Andreas Jakoby, and Till Tantau. 'Logspace versions of the theorems of Bodlaender and Courcelle'. In: *Proceedings of 51st IEEE Symposium on Foundations of Computer Science (FOCS)*. 2010, pp. 143–152.

[EK14] Michael Elberfeld and Ken-ichi Kawarabayashi. 'Embedding and canonizing graphs of bounded genus in logspace'. In: *Proceedings of 46th Annual ACM Symposium on Theory of Computing (STOC)*. New York: ACM, 2014, pp. 383–392.

[EPT00] Sergei A. Evdokimov, Ilya N. Ponomarenko, and Gottfried Tinhofer. 'Forestal algebras and algebraic forests (on a new class of weakly compact graphs)'. In: *Discrete Mathematics* 255.1-3 (Oct. 2000), pp. 149–172.

[ES93] Elaine M. Eschen and Jeremy P. Spinrad. 'An $O(n^2)$ algorithm for circular-arc graph recognition'. In: *Proceedings of 4th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. Philadelphia, PA, USA: SIAM, 1993, pp. 128–137.

[EST12] Michael Elberfeld, Christoph Stockhusen, and Till Tantau. 'On the space complexity of parameterized problems'. In: *Proceedings of 7th International Symposium on Parameterized and Exact Computation (IPEC)*. LNCS 7535. Berlin: Springer, 2012, pp. 206–217.

[Ete97] Kousha Etessami. 'Counting quantifiers, successor relations, and logarithmic space'. In: *Journal of Computer and System Sciences* 54.3 (June 1997), pp. 400–411.

[FG03] Jörg Flum and Martin Grohe. 'Describing parameterized complexity classes'. In: *Information and Computation* 187.2 (Dec. 2003), pp. 291–319.

[FG65] Delbert Ray Fulkerson and Oliver Gross. 'Incidence matrices and interval graphs'. In: *Pacific Journal of Mathematics* 15.3 (Nov. 1965), pp. 835–855.

[FHL80] Merrik Furst, John Edward Hopcroft, and Eugene M. Luks. 'Polynomial-time algorithms for permutation groups'. In: *Proceedings of 21st IEEE Symposium on Foundations of Computer Science (SFCS)*. IEEE, 1980, pp. 36–41.

[Gat79] Georg Gati. 'Further annotated bibliography on the isomorphism disease'. In: *Journal of Graph Theory* 3 (1979), pp. 95–109.

[Gav74] Fanica Gavril. 'Algorithms on circular-arc graphs'. In: *Networks* 4 (1974), pp. 357–369.

[GG14] Andrew Gainer-Dewar and Ira M. Gessel. 'Counting unlabeled *k*-trees'. In: *Journal of Combinatorial Theory*. A 126 (Aug. 2014), pp. 177–193.

[GM12] Martin Grohe and Dániel Marx. 'Structure theorem and isomorphism test for graphs with excluded topological subgraphs'. In: *Proceedings of 44th Annual ACM Symposium on Theory of Computing (STOC)*. 2012, pp. 173–192.

[Gol04] Martin Charles Golumbic. *Algorithmic graph theory and perfect graphs*. 2nd ed. Annals of Discrete Mathematics 57. Amsterdam: Elsevier, 2004. ISBN: 978-0-444-51530-8.

[GPZ08] Fanica Gavril, Ron Y. Pinter, and Shmuel Zaks. 'Intersection representations of matrices by subtrees and unicycles on graphs'. In: *Journal of Discrete Algorithms* 6.2 (June 2008), pp. 216–228.

[GS86] Shafi Goldwasser and Michael Sipser. 'Private coins versus public coins in interactive proof systems'. In: *Proceedings of 18th Annual ACM Symposium on Theory of Computing (STOC)*. 1986, pp. 59–68.

[GSS02] John G. Del Greco, Chandra N. Sekharan, and R. Sridhar. 'Fast parallel reordering and isomorphism testing of *k*-trees'. In: *Algorithmica* 32.1 (2002).

[GV06] Martin Grohe and Oleg Verbitsky. 'Testing graph isomorphism in parallel by playing a game'. In: *Proceedings of 33rd International Colloquium on Automata, Languages and Programming (ICALP)*. LNCS 4051. Berlin: Springer, 2006, pp. 3–14.

[HAB02] William Hesse, Eric Allender, and David A. Mix Barrington. 'Uniform constant-depth threshold circuits for division and iterated multiplication'. In: *Journal of Computer and System Sciences* 65.4 (Dec. 2002), pp. 695–716.

[HM03] Wen-Lian Hsu and Ross M. McConnell. 'PC trees and circular-ones arrangements'. In: *Theoretical Computer Science* 296.1 (Mar. 2003), pp. 99–116.

[HM96] Frank Harary and Terry A. McKee. 'The square of a chordal graph'. In: *Discrete Mathematics* 128.1-3 (Apr. 1996), pp. 165–172.

[HM99] Wen-Lian Hsu and Tze-Heng Ma. 'Fast and simple algorithms for recognizing chordal comparability graphs and interval graphs'. In: *SIAM Journal on Computing* 28.3 (1999), pp. 1004–1020.

[HMP+00] Michel Habib, Ross McConnell, Christophe Paul, and Laurent Viennot. 'Lex-BFS and partition refinement, with applications to transitive orientation, interval graph recognition and consecutive ones testing'. In: *Theoretical Computer Science* 234.1-2 (Mar. 2000), pp. 59–84.

[HMR93] Michel Habib, Michel Morvan, and Jean-Xavier Rampon. 'On the calculation of transitive reduction—closure of orders'. In: *Discrete Mathematics* 111.1–3 (Feb. 1993), pp. 289–303.

[Hof82] Christoph Martin Hoffmann. *Group-theoretic algorithms and graph isomorphism*. LNCS 136. Berlin: Springer, 1982. ISBN: 978-3-540-11493-2.

[HSS01] Pavol Hell, Ron Shamir, and Roded Sharan. 'A fully dynamic algorithm for recognizing and representing proper interval graphs'. In: *SIAM Journal on Computing* 31.1 (2001), pp. 289–305.

[Hsu02] Wen-Lian Hsu. 'A simple test for the consecutive ones property'. In: *Journal of Algorithms* 43.1 (Apr. 2002), pp. 1–16.

[Hsu95] Wen-Lian Hsu. '$O(m \cdot n)$ algorithms for the recognition and isomorphism problems on circular-arc graphs'. In: *SIAM Journal on Computing* 24.3 (1995), pp. 411–439.

[HT71] John Edward Hopcroft and Robert Tarjan. 'A V^2 algorithm for determining isomorphism of planar graphs'. In: *Information Processing Letters* 1 (1971), pp. 32–34.

[HT72] John Edward Hopcroft and Robert Tarjan. 'Isomorphism of Planar Graphs'. In: *Complexity of Computer Computations*. 1972, pp. 131–152.

[Hua95] Jing Huang. 'On the structure of local tournaments'. In: *Journal of Combinatorial Theory*. B 63.2 (Mar. 1995), pp. 200–221.

[HW74] John Edward Hopcroft and J. K. Wong. 'Linear time algorithm for isomorphism of planar graphs'. In: *Proceedings of 6th Annual ACM Symposium on Theory of Computing (STOC)*. 1974, pp. 172–184.

[Imm87] Neil Immerman. 'Languages which capture complexity classes'. In: *SIAM Journal on Computing* 16.4 (1987), pp. 760–778.

[Imm88] Neil Immerman. 'Nondeterministic space is closed under complementation'. In: *SIAM Journal on Computing* 17.5 (1988), pp. 935–938.

[JKM+03] Birgit Jenner, Johannes Köbler, Pierre McKenzie, and Jacobo Torán. 'Completeness results for graph isomorphism'. In: *Journal of Computer and System Sciences* 66.3 (May 2003), pp. 549–566. See also [JKM+06].

[JKM+06] Birgit Jenner, Johannes Köbler, Pierre McKenzie, and Jacobo Torán. 'Corrigendum to "Completeness results for graph isomorphism"'. In: *Journal of Computer and System Sciences* 72.4 (June 2006), p. 783. Corrects a proof in [JKM+03].

[JLM+11] Benson L. Joeris, Min Chih Lin, Ross M. McConnell, Jeremy P. Spinrad, and Jayme Luiz Szwarcfiter. 'Linear time recognition of Helly circular-arc models and graphs'. In: *Algorithmica* 59.2 (Feb. 2011), pp. 215–239.

[KCP82] Maria M. Klawe, Derek G. Corneil, and Andrzej Proskurowski. 'Isomorphism testing in hookup classes'. In: *SIAM Journal on Algebraic and Discrete Methods* 3.2 (June 1982), pp. 260–274.

[KK09] Johannes Köbler and Sebastian Kuhnert. 'The isomorphism problem for k-trees is complete for logspace'. In: *Proceedings of 34th International Symposium on Mathematical Foundations of Computer Science (MFCS)*. LNCS 5734. Berlin: Springer, 2009, pp. 537–548.

[KKL+10] Johannes Köbler, Sebastian Kuhnert, Bastian Laubner, and Oleg Verbitsky. 'Interval graphs: Canonical representation in logspace'. In: *Proceedings of 37th International Colloquium on Automata, Languages and Programming (ICALP)*. LNCS 6198. Berlin: Springer, 2010, pp. 384–395.

[KKL+11] Johannes Köbler, Sebastian Kuhnert, Bastian Laubner, and Oleg Verbitsky. 'Interval graphs: Canonical representations in logspace'. In: *SIAM Journal on Computing* 40.5 (2011), pp. 1292–1315.

[KKV11] Pavel Klavík, Jan Kratochvíl, and Tomáš Vyskočil. 'Extending partial representations of interval graphs'. In: *Proceedings of 8th Annual Conference on Theory and Applications of Models of Computation (TAMC)*. LNCS 6648. Berlin: Springer, 2011, pp. 276–285.

[KKV12a] Johannes Köbler, Sebastian Kuhnert, and Oleg Verbitsky. 'Around and beyond the isomorphism problem for interval graphs'. In: *Bulletin of the European Association for Theoretical Computer Science* 107 (June 2012). Computational Complexity Column, pp. 44–71.

[KKV12b] Johannes Köbler, Sebastian Kuhnert, and Oleg Verbitsky. 'Solving the canonical representation and star system problems for proper circular-arc graphs in logspace'. In: *Proceedings of 32nd Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*. LIPIcs 18. Dagstuhl: Leibniz-Zentrum für Informatik, 2012, pp. 387–399.

[KKV13a] Johannes Köbler, Sebastian Kuhnert, and Oleg Verbitsky. 'Helly circular-arc graph isomorphism is in logspace'. In: *Proceedings of 38th International Symposium on Mathematical Foundations of Computer Science (MFCS)*. LNCS 8087. Berlin: Springer, 2013, pp. 631–642.

[KKV13b] Johannes Köbler, Sebastian Kuhnert, and Oleg Verbitsky. *Solving the canonical representation and star system problems for proper circular-arc graphs in logspace*. Dec. 5, 2013. arXiv: 1202.4406v5.

[KKV14] Johannes Köbler, Sebastian Kuhnert, and Oleg Verbitsky. *On the isomorphism problem for Helly circular-arc graphs*. Feb. 19, 2014. arXiv: 1402.4642v1.

[KKW12] Johannes Köbler, Sebastian Kuhnert, and Osamu Watanabe. 'Interval graph representation with given interval and intersection lengths'. In: *Proceedings of 23rd International Symposium on Algorithms and Computation (ISAAC)*. LNCS 7676. Berlin: Springer, 2012, pp. 517–526.

[Kle96] Philip N. Klein. 'Efficient parallel algorithms for chordal graphs'. In: *SIAM Journal on Computing* 25.4 (1996), pp. 797–827.

[Klo94] Ton Kloks. *Treewidth. Computations and approximations*. LNCS 842. Berlin: Springer, 1994. ISBN: 3-540-58356-4.

[KM89] Norbert Korte and Rolf H. Möhring. 'An incremental linear-time algorithm for recognizing interval graphs'. In: *SIAM Journal on Computing* 18.1 (Feb. 1989), pp. 68–81.

[KN09] Haim Kaplan and Yahav Nussbaum. 'Certifying algorithms for recognizing proper circular-arc graphs and unit circular-arc graphs'. In: *Discrete Applied Mathematics* 157.15 (Aug. 2009), pp. 3216–3230.

[KN11] Haim Kaplan and Yahav Nussbaum. 'A simpler linear-time recognition of circular-arc graphs'. In: *Algorithmica* 61.3 (2011), pp. 694–737.

[Köb06] Johannes Köbler. 'On graph isomorphism for restricted graph classes'. In: *Logical Approaches to Computational Barriers. Proceedings of 2nd Conference on Computability in Europe (CiE)*. LNCS 3988. Berlin: Springer, 2006, pp. 241–256.

[KR88] Philip N. Klein and John H. Reif. 'An efficient parallel algorithm for planarity'. In: *Journal of Computer and System Sciences* 37.2 (Oct. 1988), pp. 190–246.

[KST93] Johannes Köbler, Uwe Schöning, and Jacobo Torán. *The graph isomorphism problem: Its structural complexity*. Progress in Theoretical Computer Science. Boston et al.: Birkhäuser, 1993. ISBN: 0-8176-3680-3.

[KV08] Johannes Köbler and Oleg Verbitsky. 'From invariants to canonization in parallel'. In: *Proceedings of 3rd International Computer Science Symposium in Russia (CSR)*. LNCS 5010. Berlin: Springer, 2008, pp. 216–227.

[KVV85] Dexter Kozen, Umesh V. Vazirani, and Vijay V. Vazirani. 'NC algorithms for comparability graphs, interval graphs, and testing for unique perfect matching'. In: *Foundations of Software Technology and Theoretical Computer Science*. LNCS 206. Berlin: Springer, 1985, pp. 496–503.

[Lad75] Richard E. Ladner. 'On the Structure of Polynomial Time Reducibility'. In: *Journal of the ACM* 22.1 (Jan. 1975), pp. 155–171.

[Lau10] Bastian Laubner. 'Capturing polynomial time on interval graphs'. In: *Proceedings of 25th IEEE Symposium on Logic in Computer Science (LICS)*. 2010, pp. 199–208.

[Lau11] Bastian Laubner. 'The structure of graphs and new logics for the characterization of Polynomial Time'. PhD thesis. Humboldt-Universität zu Berlin, 2011.

[LB79] George S. Lueker and Kellogg S. Booth. 'A linear time algorithm for deciding interval graph isomorphism'. In: *Journal of the ACM* 26.2 (Apr. 1979), pp. 183–195.

[LB95] Y. Daniel Liang and Norbert Blum. 'Circular convex bipartite graphs: Maximum matching and Hamiltonian circuits'. In: *Information Processing Letters* 56.4 (Nov. 1995), pp. 215–219.

[Lin92] Steven Lindell. 'A logspace algorithm for tree canonization. Extended abstract'. In: *Proceedings of 24th Annual ACM Symposium on Theory of Computing (STOC)*. 1992, pp. 400–404.

[LPP+14] Daniel Lokshtanov, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Fixed-parameter tractable canonization and isomorphism test for graphs of bounded treewidth*. Apr. 3, 2014. arXiv: 1404.0818v1.

[LS08] Min Chih Lin and Jayme Luiz Szwarcfiter. 'Unit circular-arc graph representations and feasible circulations'. In: *SIAM Journal on Discrete Mathematics* 22.1 (2008), pp. 409–423.

[LS09] Min Chih Lin and Jayme Luiz Szwarcfiter. 'Characterizations and recognition of circular-arc graphs and subclasses: A survey'. In: *Discrete Mathematics* 309.18 (Sept. 2009), pp. 5618–5635.

[LSS08] Min Chih Lin, Francisco J. Soulignac, and Jayme Luiz Szwarcfiter. 'A simple linear time algorithm for the isomorphism problem on proper circular-arc graphs'. In: *Proceedings of 11th Scandinavian Workshop on Algorithm Theory (SWAT)*. LNCS 5124. Berlin: Springer, 2008, pp. 355–366.

[Luk82] Eugene M. Luks. 'Isomorphism of graphs of bounded valence can be tested in polynomial time'. In: *Journal of Computer and System Sciences* 25.1 (Aug. 1982), pp. 42–65.

[McC03] Ross M. McConnell. 'Linear-time recognition of circular-arc graphs'. In: *Algorithmica* 37.2 (2003), pp. 93–147.

[Mil80] Gary L. Miller. 'Isomorphism testing for graphs of bounded genus'. In: *Proceedings of 12th Annual ACM Symposium on Theory of Computing (STOC)*. 1980, pp. 225–235.

[Mil83] Gary L. Miller. 'Isomorphism of k-contractible graphs. A generalization of bounded valence and bounded genus'. In: *Information and Control* 56.1-2 (1983), pp. 1–20.

[MR91] Gary L. Miller and John H. Reif. 'Parallel tree contraction part 2: Further applications'. In: *SIAM Journal on Computing* 20.6 (1991), pp. 1128–1147.

[Nag01] Takayuki Nagoya. 'Counting graph isomorphisms among chordal graphs with restricted clique number'. In: *Proceedings of 12th International Symposium on Algorithms and Computation (ISAAC)*. LNCS 2223. Berlin: Springer, 2001, pp. 136–147.

[OBS11] Aïda Ouangraoua, Anne Bergeron, and Krister M. Swenson. 'Theory and practice of ultra-perfection'. In: *Journal of Computational Biology* 18.9 (Sept. 2011), pp. 1219–1230.

[OS13] Yota Otachi and Pascal Schweitzer. 'Isomorphism on subgraph-closed graph classes: A complexity dichotomy and intermediate graph classes'. In: *Proceedings of 24th International Symposium on Algorithms and Computation (ISAAC)*. LNCS 8283. Berlin: Springer, 2013, pp. 111–118.

[Pon88] Ilya N. Ponomarenko. 'The isomorphism problem for classes of graphs closed under contraction'. In: *Zapiski Nauchnykh Seminarov Leningradskogo Otdeleniya Matematicheskogo Instituta* 174 (1988), pp. 147–177. Russian. Translation to English: [Pon91].

[Pon91] Ilya N. Ponomarenko. 'The isomorphism problem for classes of graphs closed under contraction'. In: *Journal of Mathematical Sciences* 55.2 (June 1991), pp. 1621–1643.

[PS97] Itsik Pe'er and Ron Shamir. 'Realizing interval graphs with size and distance constraints'. In: *SIAM Journal on Discrete Mathematics* 10.4 (Nov. 1997), pp. 662–687.

[RC77] Ronald C. Read and Derek G. Corneil. 'The graph isomorphism disease'. In: *Journal of Graph Theory* 1 (1977), pp. 339–363.

[Rei08] Omer Reingold. 'Undirected connectivity in log-space'. In: *Journal of the ACM* 55.4 (Sept. 2008), 17:1–17:24.

[Rei84] John H. Reif. 'Symmetric complementation'. In: *Journal of the ACM* 31.2 (Apr. 1984), pp. 401–421.

[Rob69] Fred S. Roberts. 'Indifference graphs'. In: *Proof techniques in graph theory. Proceedings of 2nd Ann Arbor Graph Theory Conference*. New York: Academic Press, 1969, pp. 139–146.

[Rob71] Fred S. Roberts. 'On the compatibility between a graph and a simple order'. In: *Journal of Combinatorial Theory*. B 11.1 (Aug. 1971), pp. 28–38.

[Ros74] Donald J. Rose. 'On simple characterizations of k-trees'. In: *Discrete Mathematics* 7.3–4 (1974), pp. 317–322.

[RR96] Vijaya Ramachandran and John Reif. 'Planarity testing in parallel'. In: *Journal of Computer and System Sciences* 49.3 (Dec. 1996), pp. 517–561.

[RTL76] Donald J. Rose, R. Endre Tarjan, and George S. Lueker. 'Algorithmic aspects of vertex elimination on graphs'. In: *SIAM Journal on Computing* 5.2 (June 1976), pp. 266–283.

[Sch88] Uwe Schöning. 'Graph isomorphism is in the low hierarchy'. In: *Journal of Computer and System Sciences* 37.3 (Dec. 1988), pp. 321–323.

[Sou14] Francisco J. Soulignac. *Minimal and short representations of unit interval and unit circular-arc graphs*. Oct. 8, 2014. arXiv: 1408.3443v2.

[Spi03] Jeremy P. Spinrad. *Efficient graph representations*. Field Institute Monographs 19. AMS, 2003. ISBN: 978-0-8218-2815-1.

[Sze88] Róbert Szelepcsényi. 'The method of forced enumeration for nondeterministic automata'. In: *Acta Informatica* 26.3 (1988), pp. 279–284.

[Tod06] Seinosuke Toda. 'Computing automorphism groups of chordal graphs whose simplicial components are of small size'. In: *IEICE Transactions on Information and Systems* E89-D.8 (2006), pp. 2388–2401.

[Tor04] Jacobo Torán. 'On the hardness of graph isomorphism'. In: *SIAM Journal on Computing* 33.5 (2004), pp. 1093–1108.

[Tuc71] Alan Tucker. 'Matrix characterizations of circular-arc graphs'. In: *Pacific Journal of Mathematics* 39.2 (1971), pp. 535–545.

[Tuc72] Alan Tucker. 'A structure theorem for the consecutive 1's property'. In: *Journal of Combinatorial Theory*. B 12.2 (Apr. 1972), pp. 153–162.

[Tuc74] Alan Tucker. 'Structure theorems for some circular-arc graphs'. In: *Discrete Mathematics* 7.1-2 (1974), pp. 167–195.

[Tuc80] Alan Tucker. 'An efficient test for circular-arc graphs'. In: *SIAM Journal on Computing* 9.1 (Feb. 1980), pp. 1–24.

[Ueh08] Ryuhei Uehara. 'Simple geometrical intersection graphs'. In: *WALCOM: Algorithms and Computation*. LNCS 4921. Berlin: Springer, 2008, pp. 25–33.

[Ueh13] Ryuhei Uehara. 'Tractabilities and intractabilities on geometric intersection graphs'. In: *Algorithms* 6.1 (Jan. 2013), pp. 60–83.

[Ueh14] Ryuhei Uehara. 'The graph isomorphism problem on geometric graphs'. In: *Discrete Mathematics & Theoretical Computer Science* 16.2 (2014), pp. 87–96.

[UTN05] Ryuhei Uehara, Seinosuke Toda, and Takayuki Nagoya. 'Graph isomorphism completeness for chordal bipartite graphs and strongly chordal graphs'. In: *Discrete Applied Mathematics* 145.3 (2005), pp. 479–482.

[Wag11] Fabian Wagner. 'Graphs of bounded treewidth can be canonized in AC^1'. In: *Proceedings of 6th International Computer Science Symposium in Russia (CSR)*. LNCS 6651. Berlin: Springer, 2011, pp. 209–222.

[Wan94] Egon Wanke. 'Bounded tree-width and LOGCFL'. In: *Journal of Algorithms* 16.3 (May 1994), pp. 470–491.

[Weg67] Gerd Wegner. »Eigenschaften der Nerven homologisch-einfacher Familien im $\mathbb{R}^n$«. Diss. Universität Göttingen, 1967.

[Yam07] Naoki Yamamoto. 'Weighted interval graphs and their representations'. Japanese. Master's Thesis. Tokyo Inst. of Technology, 2007.

[YBF+99] Koichi Yamazaki, Hans L. Bodlaender, Babette de Fluiter, and Dimitrios M. Thilikos. 'Isomorphism for graphs of bounded distance width'. In: *Algorithmica* 24.2 (June 1999), pp. 105–127.

[YC96] Chang-Wu Yu and Gen-Huey Chen. 'An efficient parallel recognition algorithm for bipartite-permutation graphs'. In: *IEEE Transactions on Parallel and Distributed Systems* 7.1 (Jan. 1996), pp. 3–10.

[Zem70] Viktor N. Zemlyachenko. 'Canonical numbering of trees'. Russian. In: *Proc. Seminar on Comb. Anal. at Moscow State University*. 1970, p. 55.

[ZKT82] Viktor N. Zemlyachenko, Nikolay M. Kornienko, and Regina I. Tyshkevich. 'Graph isomorphism problem'. In: *Zapiski Nauchnykh Seminarov Leningradskogo Otdeleniya Matematicheskogo Instituta* 118 (1982), pp. 83–158. Russian. Translation to English: [ZKT85].

[ZKT85] Viktor N. Zemlyachenko, Nikolay M. Kornienko, and Regina I. Tyshkevich. 'Graph isomorphism problem'. In: *Journal of Mathematical Sciences* 29.4 (1985), pp. 1426–1481. English translation of [ZKT82].

Glossary

Acronyms

CA circular-arc.

DCIG distance constrained interval graph; see Section 4.5.4.

FPT fixed parameter tractable.
FRP fast reordering problem; see page 32.

HCA Helly circular-arc.

PCA proper circular-arc.
PEO perfect elimination order.

UCO unique clique order.

Concepts

arc model of G An arc system $\mathcal{A}$ whose intersection graph $\mathbb{I}(\mathcal{A})$ is isomorphic to the graph G.
arc model of $\mathcal{H}$ An arc system $\mathcal{A}$ that is isomorphic to the hypergraph $\mathcal{H}$.
arc model of μ An arc system $\mathcal{A}$ whose intersection matrix $\mu_{\mathcal{A}}$ is isomorphic to the matrix μ.
arc over $(X, \prec)$ A subset of X that consists of consecutive points w.r.t. the circular order $\prec$ on X.
arc representation of G An isomorphism $\alpha\colon \mathrm{V}(G) \to \mathcal{A}$ from G to the intersection graph $\mathbb{I}(\mathcal{A})$ of an arc model $\mathcal{A}$ of G.
arc representation of $\mathcal{H}$ An isomorphism $\rho\colon \mathrm{V}(\mathcal{H}) \to \mathrm{V}(\mathcal{A})$ from the hypergraph $\mathcal{H}$ to an arc model $\mathcal{A}$ of $\mathcal{H}$.
arc representation of $\mu = (\mu_{u,v})_{u \neq v \in V}$ An isomorphism $\alpha\colon V \to \mathcal{A}$ from the matrix μ to the intersection matrix $\mu_{\mathcal{A}}$ of an arc model $\mathcal{A}$ of μ.
arc system over $(X, \prec)$ A multiset of arcs over $(X, \prec)$.
automorphism An isomorphism from a (hyper)graph to itself.

base The initial k-clique in the iterative construction of a k-tree.
biconvex graphs The class of bipartite graphs whose open neighborhood hypergraphs is interval.

bundle hypergraph of G The hypergraph $\mathcal{B}(G)$ that has one node for each maxclique of the graph G and one hyperedge for the maxclique bundle of each vertex, i.e., $\{\!\{ \mathrm{B}_v \mid v \in \mathrm{V}(G) \}\!\}$.

bundle of v in G The set B_v of all maxcliques of the graph G that contain the vertex v.

CA graphs The class of graphs that are isomorphic to the intersection graph of some arc system.

CA hypergraphs The class of hypergraphs that are isomorphic to some arc system.

CA matrix A matrix that is isomorphic to the intersection matrix $\mu_{\mathcal{A}}$ of some arc system $\mathcal{A}$.

CA order of $\mathcal{H}$ A circular order $\prec$ on $\mathrm{V}(\mathcal{H})$ such that every hyperedge $A \in \mathcal{H}$ consists of consecutive points w.r.t. $\prec$.

canonical form A function f computes canonical forms for a class C of (hyper)graphs if it maps each $G \in \mathsf{C}$ to an isomorphic copy $f(G)$ of G, and isomorphic (hyper)graphs $G \cong H$ are mapped to the same canonical form $f(G) = f(H)$.

canonical labeling An isomorphism from a (hyper)graph to its canonical form.

caterpillars The class of graphs that become paths after removing all vertices of degree 1.

center The center of a graph G is the set of all vertices that have minimum eccentricity.

chordal graphs The class of graphs that do not have induced cycles of length more than 3.

circular order on X A circular successor relation, i.e., a directed cycle on X.

circular-convex graphs The class of bipartite graphs whose open neighborhood hypergraphs restricted to one vertex class is CA.

circular-ones property A boolean matrix A has the circular-ones property if it is the incidence matrix of an CA hypergraph, i.e., if its columns can be ordered so that, in every row, all ones or all zeroes appear consecutively.

closed neighborhood of v in G The set $\mathrm{N}_G[v]$ of vertices with distance at most 1 to v in the graph G.

coloring of G A (vertex) coloring of a (hyper)graph G is a function $c\colon \mathrm{V}(G) \to C$; see also *k-coloring*.

comparability graphs The class of undirected graphs whose edges can be oriented in a way that results in a partial order.

complete invariant An invariant f that maps nonisomorphic (hyper)graphs to different values, i.e., $G \cong H \Leftrightarrow f(G) = f(H)$.

concave-round graphs The class of graphs G whose neighborhood hypergraph $\mathcal{N}[G]$ is CA.

consecutive-ones property A boolean matrix A has the consecutive-ones property if it is the incidence matrix of an interval hypergraph, i.e., if its columns can be ordered so that, in every row, all ones appear consecutively.

convex graphs The class of bipartite graphs whose open neighborhood hypergraphs restricted to one vertex class is interval.

convex-round graphs The class of graphs G for which the open neighborhood hypergraph $\mathcal{N}(G)$ is CA.

dual hypergraph of $\mathcal{H}$ The hypergraph $\mathcal{H}^{\mathsf{D}} = \{\!\{ v^* \mid v \in \mathrm{V}(\mathcal{H}) \}\!\}$ with the vertex set $\mathrm{V}(\mathcal{H}^{\mathsf{D}}) = \mathcal{H}$, where $v^* = \{\!\{ A \in \mathcal{H} \mid v \in A \}\!\}$.

eccentricity The eccentricity of a vertex v in a graph G is the longest distance to another vertex; it is denoted by $\mathrm{ecc}_G(v) = \max\{\mathrm{d}_G(v,u) \mid u \in \mathrm{V}(G)\}$.

edge-coloring of $\mathcal{H}$ An edge-coloring of a hypergraph $\mathcal{H}$ is a function $c\colon \mathcal{H} \to C$.

first-order translation A many-one reduction where the output structure is defined by first-order formulas over the relations and constants of the input structure, the $=$ relation, the successor relation $\prec$ and the constant symbols min and max for the smallest and largest element of the structure w.r.t. $\prec$.

fraternal vertices Two vertices u and v of a graph G with the same neighborhood $\mathrm{N}_G(u) = \mathrm{N}_G(v)$.

geometric order Given an interval representation $\alpha\colon \mathrm{V}(G) \to \mathcal{I}$, the geometric order $<_\alpha$ on $\mathrm{V}(G)$ places $u <_\alpha v$ if $\alpha(u) < \alpha(v)$. Similarly, given an arc representation $\alpha\colon \mathrm{V}(G) \to \mathcal{A}$, the circular geometric order $\prec_\alpha$ on $\mathrm{V}(G)$ places $u \prec_\alpha v$ if $\alpha(u) \prec_{\mathcal{A}} \alpha(v)$, where $\prec_{\mathcal{A}}$ is the lifted order on the arcs of $\mathcal{A}$.

HCA graphs The class of graphs that are isomorphic to the intersection graph of some Helly arc system.

HCA matrix A matrix that is isomorphic to the intersection matrix $\mu_{\mathcal{A}}$ of some Helly arc system $\mathcal{A}$.

Helly property A hypergraph $\mathcal{H}$ is Helly if any subset $\mathcal{K} \subseteq \mathcal{H}$ with nonempty pairwise intersections $A \cap B$, for $A, B \in \mathcal{K}$, also has nonempty overall intersection $\bigcap_{A\in\mathcal{K}} A$.

incidence graph The incidence graph of a hypergraph $\mathcal{H}$ is the bipartite graph with vertex classes $\mathrm{V}(\mathcal{H})$ and $\mathcal{H}$ where two vertices $v \in \mathrm{V}(\mathcal{H})$ and $A \in \mathcal{H}$ are adjacent if $v \in A$.

incidence matrix The incidence matrix of a hypergraph $\mathcal{H}$ has one column for each vertex and one row for each hyperedge. The entries are 1 if the respective vertex is contained in the respective hyperedge, and 0 otherwise.

intersection graph of $\mathcal{H}$ The graph $\mathbb{I}(\mathcal{H})$ with vertex set $\mathcal{H}$ where $A, B \in \mathcal{H}$ are adjacent if and only if $A \cap B \neq \emptyset$.

intersection matrix of $\mathcal{A}$ The matrix $\mu_{\mathcal{A}} = (\mu_{A,B})_{A\neq B\in\mathcal{A}}$ that describes the relation between the arcs of the arc system $\mathcal{A}$, i.e., $\mu_{A,B} = \mathtt{di}$ if $A \cap B = \emptyset$, $mu_{A,B} = \mathtt{cd}$ if $A \subsetneq B$, $\mu_{A,B} = \mathtt{cs}$ if $A \supsetneq B$, $\mu_{A,B} = \mathtt{ov}$ if $A \pitchfork B$, and $\mu_{A,B} = \mathtt{cc}$ if $A \circledcirc B$.

intersection model of G A hypergraph $\mathcal{H}$ whose intersection graph $\mathbb{I}(\mathcal{H})$ is isomorphic to the graph G.

intersection representation of G An isomorphism $\alpha\colon \mathrm{V}(G) \to \mathcal{H}$ from G to the intersection graph $\mathbb{I}(\mathcal{H})$ of some hypergraph $\mathcal{H}$.

interval graphs The class of graphs that are isomorphic to the intersection graph of some interval system.

interval hypergraphs The class of hypergraphs that are isomorphic to some interval system.

interval matrix A matrix that is isomorphic to the intersection matrix $\mu_{\mathcal{I}}$ of some interval system $\mathcal{I}$.

interval model of G An interval system $\mathcal{I}$ whose intersection graph $\mathbb{I}(\mathcal{I})$ is isomorphic to the graph G.

interval model of $\mathcal{H}$ An interval system $\mathcal{I}$ that is isomorphic to the hypergraph $\mathcal{H}$.

interval model of μ An interval system $\mathcal{I}$ whose intersection matrix $\mu_{\mathcal{I}}$ is isomorphic to the matrix μ.

interval order of $\mathcal{H}$ An order $\leq$ on the vertices of the interval hypergraph $\mathcal{H}$ such that $v \mapsto |\{u \mid u \leq v\}|$ is an interval representation of $\mathcal{H}$.

interval representation of G An isomorphism $\alpha\colon \mathrm{V}(G) \to \mathcal{I}$ from G to the intersection graph $\mathbb{I}(\mathcal{I})$ of an interval model $\mathcal{I}$ of G.

interval representation of $\mathcal{H}$ An isomorphism $\rho\colon \mathrm{V}(\mathcal{H}) \to \mathrm{V}(\mathcal{I})$ from the hypergraph $\mathcal{H}$ to an interval model $\mathcal{I}$ of $\mathcal{H}$.

interval representation of $\mu = (\mu_{u,v})_{u \neq v \in V}$ An isomorphism $\alpha\colon V \to \mathcal{I}$ from the matrix μ to the intersection matrix $\mu_{\mathcal{I}}$ of an interval model $\mathcal{I}$ of μ.

interval system A multiset of intervals over $\mathbb{N}^+$.

invariant A function f defined on some (hyper)graph class that maps isomorphic (hyper)graphs to the same value, i.e., $G \cong H \Rightarrow f(G) = f(H)$; see also *complete invariant*.

isomorphic Two (hyper)graphs are isomorphic if there is an isomorphism between them.

isomorphism A bijection between the vertex sets of two (hyper)graphs, that maps edges to edges and non-edges to non-edges, and that preserves edge-multiplicities and colors of vertices and edges.

isomorphism-complete A graph class C is isomorphism-complete if GI can be reduced to the isomorphism problem for C.

isomorphism-tractable A graph class C is isomorphism-tractable if there is a polynomial-time algorithm for the isomorphism problem for C.

k-coloring of G A k-coloring of a (hyper)graph G is a coloring $c\colon \mathrm{V}(G) \to C$ of G with $k = |C|$ that satisfies $c(u) \neq c(v)$ for any two vertices $u \neq v$ that occur together in some (hyper)edge of G.

k-paths The subclass of k-trees where the support of each vertex either includes the previously added vertex u or is the same as the support of u; see page 20.

k-trees The class of all graphs that can be obtained from the inductive construction that starts with a k-clique (called base) and allows to introduce a new vertex v if it is connected to all vertices of a previously present k-clique (called support of v).

k-uniform A hypergraph $\mathcal{H}$ is k-uniform if all its hyperedges have size k.

labeling of G A labeling of a (hyper)graph G is a bijection $\ell\colon \mathrm{V}(G) \to \{1, \ldots, |\mathrm{V}(G)|\}$.

lifted order Given a CA order $\prec$ of a hypergraph $\mathcal{H}$ with $\emptyset, \mathrm{V}(\mathcal{H}) \notin \mathcal{H}$, the lifted order $\prec_{\mathcal{H}}$ is the circular order on $\mathcal{H}$ that makes an arc $B \in \mathcal{H}$ the successor of $A \in \mathcal{H}$ if there is no arc $C \in \mathcal{H}$ that occurs between A and B in the circular order $\prec^*$ of all nonempty and nonuniversal arcs, where $[a^-, a^+] \prec^* [b^-, b^+]$ if $a^- = b^-$ and $a^+ \prec b^+$ or if $a^+ \prec a^- \prec b^- = b^+$.

ℓ-respecting Given a graph G and a function $\ell\colon \mathrm{V}(G) \to \mathbb{N}^+$, an interval representation ρ of G is ℓ-respecting if $|\rho(v)| = \ell(v)$ for all $v \in \mathrm{V}(G)$.

(ℓ, s)-respecting Given a graph G and functions $\ell\colon \mathrm{V}(G) \to \mathbb{N}^+$ and $s\colon \mathrm{V}(G) \to \mathbb{N}^+$, an interval representation ρ of G is (ℓ, s)-respecting if $|\rho(v)| = \ell(v)$ and $|\rho(e)| = s(e)$ for all $v \in \mathrm{V}(G)$ and all $e \in \mathrm{E}(G)$, respectively.

maxclique hypergraph of G The hypergraph $\mathcal{C}(G)$ that has the same vertex set as the graph G and one hyperedge for each maxclique of G.

maxclique of G An clique of the graph G that is not contained in a larger clique.

neighborhood hypergraph of G The hypergraph $\mathcal{N}[G]$ with the same vertex set as G and the hyperedges $\{\!\{\mathrm{N}[v] \mid v \in \mathrm{V}(G)\}\!\}$; also called *closed neighborhood hypergraph* to contrast with the open neighborhood hypergraph.

neighborhood matrix of G See Definition 5.2.2.

neighborhood of v in G The set $\mathrm{N}_G(v)$ of vertices with distance 1 to v in the graph G; also called *open neighborhood* when contrasted against the closed neighborhood.

open neighborhood hypergraph of G The hypergraph $\mathcal{N}(G)$ with the same vertex set as G and the hyperedges $\{\!\{\mathrm{N}(v) \mid v \in \mathrm{V}(G)\}\!\}$; see also *neighborhood hypergraph*.

***O*-respecting** Given an CA matrix $\mu = (\mu_{u,v})_{u \neq v \in V}$ and a family of equivalence relations $O = (O_u)_{u \in V}$, an arc representation α of μ is O-respecting if for any three vertices $u, v, w \in \mathrm{V}(G)$ with $\mu_{u,v} = \mu_{u,w} = \mathsf{ov}$, the arcs $\alpha(v)$ and $\alpha(w)$ contain the same extreme point of $\alpha(u)$ if and only if $(v, w) \in O_u$.

overlap Two sets A and B overlap (denoted $A \between B$) if they have nonempty intersection and neither contains the other.

overlap component of $\mathcal{H}$ A subhypergraph $\mathcal{O} \subseteq \mathcal{H}$ whose hyperedges correspond to a connected component of the overlap graph $\mathbb{O}(\mathcal{H})$.

overlap component tree of $\mathcal{H}$ The rooted tree that has the overlap components of the connected hypergraph $\mathcal{H}$ as its vertices, where $\mathcal{O}$ is in the subtree rooted at $\mathcal{O}'$ if and only if there are $A \in \mathcal{O}$ and $B \in \mathcal{O}'$ such that $A \subseteq B$.

overlap graph of $\mathcal{H}$ The graph $\mathbb{O}(\mathcal{H})$ with vertex set $\mathcal{H}$ where $A, B \in \mathcal{H}$ are adjacent if and only if $A \between B$ or $A = B$.

PCA graphs The class of graphs that are isomorphic to the intersection graph of some proper arc system.

permutation graphs The class of graphs that are isomorphic to the intersection graphs of straight lines between two parallel lines.

PQ-tree A rooted ordered tree whose inner nodes are labelled either with P or with Q. The children of a P node may be reordered arbitrarily, while the order of the children of a Q node may only be reversed. PQ-trees can be used to find interval representations.

proper A hypergraph $\mathcal{H}$ is proper if no hyperedge of $\mathcal{H}$ contains another.

proper interval graphs The class of graphs that are isomorphic to the intersection graph of some proper interval system.

quotient graph of G The graph G' with the vertices $\mathrm{V}(G') = \{[v] \mid v \in \mathrm{V}(G)\}$ and the edges $\mathrm{E}(G') = \{\{[u], [v]\} \mid \{u, v\} \in \mathrm{E}(G)\}$, where $[v]$ is the twin class of v, i.e., $[v] = \{w \in \mathrm{V}(G) \mid \mathrm{N}[v] = \mathrm{N}[w]\}$.

rigid A (hyper)graph is rigid if its only automorphism is the identity.

sharp arc system A system $\mathcal{A}$ of m arcs on the circle $\mathbb{C}_{2m}$ is sharp if every point $x \in \mathbb{C}_{2m}$ is the extreme point of exactly one arc.

simplicial A vertex v of a graph G is simplicial if its neighborhood $\mathrm{N}_G(v)$ is a clique.

slot of $\mathcal{H}$ An inclusion-maximal subset S of the vertices of the hypergraph $\mathcal{H}$ such that each hyperedge $A \in \mathcal{H}$ contains either all of S or none of it.

***s*-respecting** Given a graph G and a function $s\colon \mathrm{V}(G) \to \mathbb{N}^+$, an interval representation ρ of G is s-respecting if $|\rho(e)| = s(e)$ for all $e \in \mathrm{E}(G)$.

strict overlap Two arcs A and B on a circle $\mathbb{C}$ strictly overlap ($A \curvearrowright B$) if they overlap (i.e., $A \between B$) and each contains only one extreme point of the other.

support of $\mathcal{H}$ The set of non-isolated vertices of the hypergraph $\mathcal{H}$, i.e., $\bigcup_{A \in \mathcal{H}} A$.

support of v The k-clique to which the vertex v is connected during the iterative construction of a k-tree.

tight An arc system $\mathcal{A}$ is tight, if for any two arcs A and B with $A \subseteq B$, the difference $B \setminus A$ is also an arc.

tightened hypergraph of $\mathcal{H}$ Given a hypergraph $\mathcal{H}$, its tightened hypergraph is defined by $\mathcal{H}^{\Subset} = \mathcal{H} \cup \{A \setminus B \mid A, B \in \mathcal{H}, B \subset A\}$.

trapezoid graphs The class of graphs that are isomorphic to the intersection graphs of trapezoids between two parallel lines.

tree decomposition of G A tree $T = (V_T, E_T)$ where $V_T \subseteq \mathcal{P}\big(\mathrm{V}(G)\big)$, such that (a) for each edge $\{u, v\} \in \mathrm{E}(G)$, there is a node $M \in V_T$ with $\{u, v\} \subseteq M$, and (b) for every vertex $v \in \mathrm{V}(G)$, the nodes of T that contain v induce a nonempty subtree of T.

treewidth of G The smallest $k \in \mathbb{N}$ such that G admits a tree decomposition T of width k, i.e., where all nodes of T have size at most $k + 1$.

twins of G Two vertices u and v of the graph G with the same closed neighborhood $\mathrm{N}_G[u] = \mathrm{N}_G[v]$.

twins of $\mathcal{H}$ Two vertices u and v of the hypergraph $\mathcal{H}$ such that every hyperedge $A \in \mathcal{H}$ contains either both or none of them.

UCO graphs The class of interval graphs whose bundle hypergraph has a unique interval representation (up to reversing); see Section 4.5.4.

unit An arc system $\mathcal{A}$ is unit if all its arcs have the same length.

universal vertex A vertex v of a graph G is universal if it is adjacent to all other vertices, i.e., if $\mathrm{N}_G[v] = \mathrm{V}(G)$.

Notations

$A \between B$ True if the sets A and B overlap, i.e., all of $A \cap B$, $A \setminus B$ and $B \setminus A$ are nonempty.

$A \curvearrowright B$ True if the arcs A and B of a circle $\mathbb{C}$ strictly overlap, i.e., $A \between B$ and A contains only one extreme point of B.

$A \circlearrowright B$ True if two arcs $A = [a^-, a^+]$ and $B = [b^-, b^+]$ jointly cover the circle, i.e., if $A \between B$ and $b^-, b^+ \in A$.

AC^k The class of functions computable by uniform families of polynomial-size depth $O\big(\log^k(n)\big)$ boolean circuits with unbounded fanin *and*, *or*, and *not* gates.

$A \triangle B$ The symmetric difference of two sets A and B.

A^{F} The flipped version $[b, a]$ of the arc $A = [a, b]$.

AM The complexity class of all problems that admit an Arthur-Merlin protocol.

$\mathrm{Aut}(G)$ The automorphism group of the (hyper)graph G.

$\mathcal{B}(G)$ The bundle hypergraph of the graph G, i.e., $\{\!\{\mathrm{B}_v \mid v \in \mathrm{V}(G)\}\!\}$.
B_v The maxclique bundle of the vertex v, i.e., the set of all maxcliques that contain v.

$\mathcal{C}(G)$ The maxclique hypergraph of the graph G.
$\mathbb{C}_n$ The set $\{1, \dots, n\}$ together with the circular order $1 \prec 2 \prec \cdots \prec n \prec 1$.
C_n The undirected circle on n vertices.
coC The complexity class that consists of all problems whose complement is in C.

DCIG The problem whether a given graph G admits an interval representation where the extreme points satisfy a system of difference inequalities. The acronym stands for *distance constrained interval graph*; see Section 4.5.4 for details.
$\deg_G(v)$ The degree of the vertex v in the graph G, i.e., $|\mathrm{N}_G(v)|$.
DET The complexity class of all problems that admit an FL reduction to computing the determinant of $n \times n$ matrices of n bit integers.
$\mathrm{d}_G(u, v)$ The distance of two vertices u and v in the graph G.
DiPathCenter The set of all pairs $\langle P, c \rangle$ such that the vertex c is the center of the directed path P; this problem is L-complete under first-order translations by Lemma 2.8.1.
DLogTime The complexity class of all functions, for which each output bit can be computed in $O(\log n)$ time on a random access machine.

$\mathrm{E}(G)$ The edge set of the graph G.
$\mathrm{ecc}_G(v)$ The eccentricity of the vertex v in the graph G, i.e., $\max\{\mathrm{d}_G(v, u) \mid u \in \mathrm{V}(G)\}$.

FL The class of all functions computable by logspace transducers; see page 20.
FPT The complexity class of all parameterized problems that can be solved in $f(k)n^{O(1)}$ time, where f is a function that depends only on the parameter.

$G[U]$ The subgraph of G induced by $U \subseteq \mathrm{V}(G)$.
GA The graph automorphism problem, i.e., the set of all graphs that have an automorphism besides the identity.
GI The graph isomorphism problem, i.e., the set of all pairs of isomorphic graphs.
GI The complexity class that consists of all problems that are polynomial-time reducible to GI.
$G - U$ The subgraph of G induced by $\mathrm{V}(G) \setminus U$.

$\mathcal{H}^{\Subset}$ The tightened hypergraph defined by $\mathcal{H}^{\Subset} = \mathcal{H} \cup \{A \setminus B \mid A, B \in \mathcal{H}, B \subset A\}$.
$\mathcal{H}^{\mathsf{D}}$ The dual hypergraph of $\mathcal{H}$.

$\mathbb{I}(\mathcal{H})$ The intersection graph of $\mathcal{H}$, i.e., the graph with the vertex set $\mathcal{H}$ and the edge set $\{\{A, B\} \mid A \cap B \neq \emptyset\}$.

K_n The complete graph on n vertices.
$\mathrm{K}_{n,m}$ The complete bipartite graph whose vertex classes consist of n and m vertices, respectively.

L The complexity class of all problems that can be decided by Turing machines with logarithmic space bound.

$\mathsf{LinTime}$ The complexity class of all problems that can be decided in deterministic linear time on a random access machine.

LogCFL The class of functions computable by uniform families of polynomial-size depth $O\big(\log(n)\big)$ boolean circuits with constant fanin *and*, unbounded fanin *or*, and *not* gates.

$\mathbb{N}$ The set of nonnegative integers.

$\mathbb{N}^+$ The set of positive integers.

$\mathcal{N}(G)$ The open neighborhood hypergraph of G, i.e., $\big\{\!\!\big\{ \mathrm{N}_G(v) \bigm| v \in \mathrm{V}(G) \big\}\!\!\big\}$.

$\mathcal{N}[G]$ The (closed) neighborhood hypergraph of G, i.e., $\big\{\!\!\big\{ \mathrm{N}_G[v] \bigm| v \in \mathrm{V}(G) \big\}\!\!\big\}$.

NC^k The class of functions computable by uniform families of polynomial-size depth $O\big(\log^k(n)\big)$ boolean circuits with bounded fanin *and*, *or*, and *not* gates.

NegCycle The set of all weighted digraphs that have a negative cycle; this problem is NL-complete by Fact 4.5.8.

$\mathrm{N}_G(v)$ The (open) neighborhood of the vertex v in the graph G, i.e., the set of all vertices with distance 1 to v.

$\mathrm{N}^-_G(v)$ The predecessors of the vertex v in the digraph G, i.e., the set $\mathrm{N}^-_G(v) = \big\{u \in \mathrm{V}(G) \bigm| (u,v) \in \mathrm{E}(G)\big\}$.

$\mathrm{N}^+_G(v)$ The successors of the vertex v in the digraph G, i.e., the set $\mathrm{N}^+_G(v) = \big\{u \in \mathrm{V}(G) \bigm| (v,u) \in \mathrm{E}(G)\big\}$.

$\mathrm{N}_G[u,v]$ The common closed neighborhood of u and v, i.e., $N_G[u,v] = \mathrm{N}_G[u] \cap \mathrm{N}_G[v]$.

$\mathrm{N}_G[v]$ The closed neighborhood of the vertex v in the graph G, i.e., the set of all vertices with distance at most 1 to v.

NL The complexity class of all problems that can be decided by nondeterministic Turing machines with logarithmic space bound.

NP The complexity class of all problems that can be decided in nondeterministic polynomial time.

$O\big(f(n)\big)$ The class of positive functions that do not grow faster than $f(n)$ asymptotically, i.e., $g(n) \in O\big(f(n)\big)$ if and only if $\limsup_{n\to\infty} \frac{g(n)}{f(n)} < \infty$.

$\mathbb{O}(\mathcal{H})$ The overlap graph of $\mathcal{H}$, i.e., the graph with the vertex set $\mathcal{H}$ and the edge set $\big\{\{A,B\} \bigm| A \between B\big\}$.

Ord The set of all triples $\langle P,s,t\rangle$ such that the vertex s comes before the vertex t on the directed path P (which is given in pointer notation); this problem is L-complete [Ete97].

P The complexity class of all problems that can be decided in deterministic polynomial time.

$\mathcal{P}(S)$ The powerset of S, i.e., $\{T \mid T \subseteq S\}$.

PathCenter The set of all pairs $\langle P,c\rangle$ such that the vertex c is the center of the undirected path P; this problem is L-complete under first-order translations by Lemma 2.8.1.

SPP The complexity class of all problems for which a nondeterministic Turing machine exists that has equally many accepting and rejecting computations

for negative instances, and two more accepting than rejecting for positive instances.

StUL — The complexity class of all problems that are accepted by an NL machine that has a unique accepting configuration and at most one path of computation between any two configurations.

$\mathrm{supp}(\mathcal{H})$ — The support of the hypergraph $\mathcal{H}$, i.e., $\bigcup_{A \in \mathcal{H}} A$.

TC^k — The class of functions computable by uniform families of polynomial-size depth $O(\log^k(n))$ boolean circuits with unbounded fanin *threshold*, *and*, *or*, and *not* gates.

$\mathrm{V}(G)$ — The vertex set of the (hyper)graph G.

List of Figures

Selbständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit mit dem Titel *Space Efficient Algorithms for Graph Isomorphism and Representation* selbständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt und sie an keiner anderen Universität eingereicht habe.

Berlin, den 27. Januar 2015 Sebastian Kuhnert